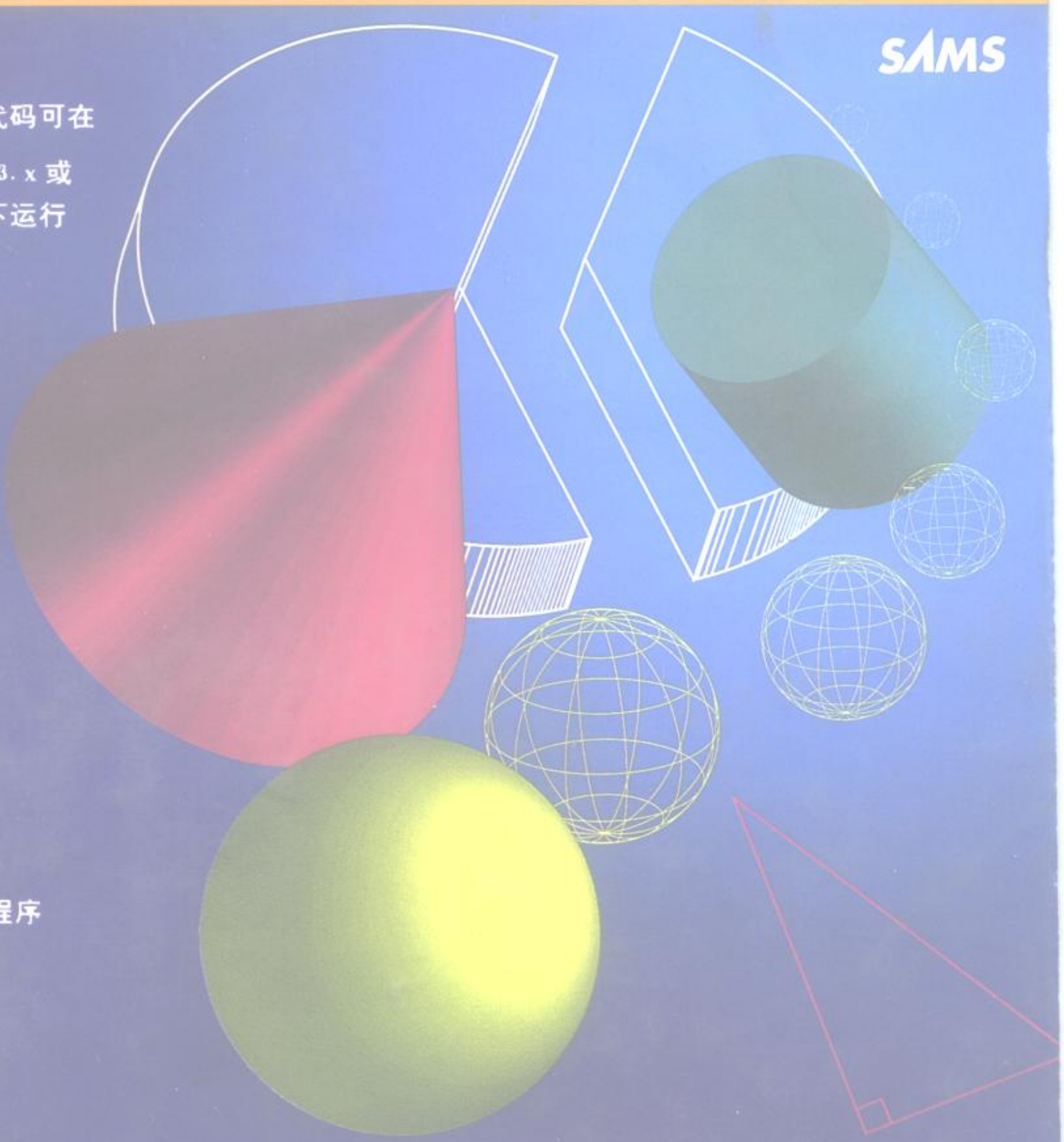
北京科海培训中心

SAMS

软盘中的源代码可在
Visual Basic 3.x 或
更高的版本下运行



- 用 Visual Basic 建立绘图工具软件
- 建立自己所需的图形程序
- 揭示绘制 Bézier 曲线、调色等关键技术

VISUAL BASIC
图形程序设计

[美] Robert W. Stewart 著
冯建华 于志宏 郑海声 译
王 华

清华大学出版社

SAMS　　北京科海培训中心

Visual Basic 图形程序设计

[illegible]图书馆

［美］ Robert W. Stewart　著
冯建华　于志宏　郑海声　译
王　华　审校

清华大学出版社

(京)新登字 158 号

JS168/16

Graphics Programming with Visual Basic

Copyright © 1995 by Sams Publishing.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher.

本书英文版由 Prentice Hall 出版社下属的 SAMS 计算机图书出版公司于 1995 年出版。版权为SAMS 公司所有。本书的中文版版权由 Prentice Hall 公司授予北京科海培训中心和清华大学出版社合作共同出版、发行。未经出版者书面允许不得以任何方式复制或抄袭本书内容。

版权所有，盗版必究。
本书封面贴有清华大学出版社激光防伪标签，无标签者不得进入各书店。

书　名：Visual Basic 图形程序设计
译　者：冯建华　于志宏　郑海声
出版者：清华大学出版社(北京清华大学校内，邮编 100084)
印刷者：北京门头沟胶印厂
发　行：新华书店总店北京科技发行所
开　本：16　印张：29.5　字数：717 千字
版　次：1995 年 10 月第 1 版　　1995 年 10 月第 1 次印刷
印　数：00001～5000
书　号：ISBN 7-302-02082-5/TP・971
定　价：59.00 元(带盘)

致谢

感谢 Grace Buechlein 在组织本书的过程中所表现的耐心和付出的努力。

感谢本书的编辑 Brad Jones 和 Angelique Brittingham 在编辑本书初稿时所作的工作，以及在编辑过程中提出的许多建议、咨询和意见。

特别要感谢 Mary Inderstrodt 和 Sandra Doell 对本书书稿的编辑工作。还应该感谢 Sams 出版社的其他成员，他们在本书的制作过程中所给予的帮助。

还应该感谢技术编辑 K. L. Murdock 确保了本书及磁盘中的所有程序都能正确执行。

最后要感谢 Sandra Guinan 在本书的初始阶段所作的编辑工作。

关于作者

Robert W. Stewart（Barrie，Ontario）大约在16年前就开始了图形艺术家的生涯。他在加拿大的广告界及报界有着丰富的工作经验。他擅长于计算机编程和桌面出版。他现在是一个独立的咨询顾问，安装基于网络的图形制作系统。Robert 居住在加拿大安大略的多伦多。

简　介

《Visual Basic 图形程序设计》包括了许多图形工具以及它们的 Visual Basic 程序。本书介绍了用以创建简单的绘图工具的代码。这些工具中的绝大部分都是用简单的 Basic 编程语句编写的。本书还介绍了怎样模拟提供给最终用户使用的绘图工具，以前这些工具只能依靠诸如C++之类的程序设计语言来产生。书中的这些程序应该被看作是起点，而不是最终产品，你可以试着将这些程序中的某些部分纳入你自己的程序。你会觉得这些例程对开发自己的 Visual Basic 程序有所裨益。

程序介绍

以下章节介绍了书中列出的所有程序的信息。

第 1 章的 Begin 程序

第 1 章介绍了几种使用 Basic 语言中 Line 和 Circle 语句的方法。这两个语句是所有的 Basic 语言都具备的标准图形函数。本章所给出的一些例子，可使你更好地理解 Line 语句的 Step 关键字以及圆和圆弧的 Start（起点）和 End（终点）位置。

第 1 章中的 Pset 程序

第 1 章还展示了 Basic 中 Pset 语句的若干例子，以及图形算法的基础知识。

第 1 章的 MoveTest 程序

使用鼠标在屏幕上移动对象的方法有许多种。本章所示的 3 个简单例程可以帮助你更好地理解其中的基本原理。

第 2 章中的 Artisan 程序

Artisan 是一个中型绘图程序，它使用了许多基本的绘图工具：形状（shape）、缩放（zoom）、线（line）、圆（circle）、方框（box）和文字（text）工具以及指针。该程序的其余部分都只是为了起装饰作用。虽然其中的控制对象可以起作用，但也都是为了增加视觉效果。例如，标尺确实可以缩放，但其中的尺寸并不是设成标准度量制。本程序中的主要部分是使用 Basic 语句绘制图形（第 15 章中的 ArtAPI 程序使用了更精致的代码及工具，并使用了 Windows API 函数）。不过，你还是应该仔细地看看 Artisan 程序，因为第 15 章中 ArtAPI 程序的许多例程都位于第 2 章的 Artsian 程序中。

Artisan 程序分布在第 1 章至第 4 章。本书首先给出声明段和各种窗体，以利于你理解每个程序中使用的变量和控制对象。因为所有的绘图操作都是在 Artwork 图片框控制对象中完成的，所以，首先给出该控制对象的各种事件过程，接着讨论 BoundingBox 过程，这些过程是对 Artwork 图片框的 MouseDown，MouseMove，MouseUp 事件的扩充。

第 3 章中的 Artisan 程序

第 3 章给出 Bézier 曲线的各种控制对象，它们用 7 个过程调用来完成计算、绘制及移动 Bézier 曲线的各项任务。当你对每个过程都很熟悉后，就可以通过组合不同的例程来缩短这些过程（尽管这样做是不必要的）。

第 4 章中的 Artisan 程序

第 4 章按字母顺序给出控制对象的事件和过程，从 C（ColorBar 图片框事件）开始，直到 N（N_H Clear 过程）结束。

第 5 章中的 Artisan 程序

第 5 章列出从 N（Node 图片框事件）到 Z（Zoom 过程）的程序和控制对象事件。

第 6 章中的 Animator 程序

本章给出一个基本公式，说明线框动画模型的基本知识。

可以给 Animator 程序加入许多功能。在熟悉了此程序后，就可以再加三个图片框控制对象，以执行生成前视图、顶视图和侧视图，或者从其他的视图平面构造线框对象。还可以将每一个单元存入位图中，然后再将这些单元串起来，以生成实时动画。Visual Basic 3.0 专业版的 PicClip VBX 控制对象可以实现该功能。

第 7 章中的多重查找和替换程序

多重查找和替换是每个字处理程序必备的内部功能。我售出的这类简单的查找替换文字编辑器比任何其他程序都要多。它使用了一个很简单的 Loop 过程，只要单击一次按钮就可以查找并替换多个单词。它是如此简单，以至于我几乎没有编入这一功能，因为我以为那些伙计们肯定在他们的字处理程序中包括进了这一基本的功能。其实不然！

第 8 章中的 API 绘图函数

如果你要在你的程序中使用 Windows API 绘图函数，你可以使用一个方便的小工具。我已将该工具写进了本书。这个工具给你展示大多数（但不是全部）API 图形调用的基本用法。请特别注意 PtInRect、FloodFill 和 Ellipse 函数。

第 9 章中的填色与调色程序

如果你有一块能够显示 256 种或更多颜色的 VGA 卡，那么第 9 章就可以让你更好地见识纯调色或喷涂填色。该章展示如何将 255％的 RGB 颜色值转换为 100％的颜色值，如何在一个调色板内显示各种颜色，如何在两种颜色之间产生颜色阴影，如何创建可显示多于 256 种自定义纯色或抖色的调色板，如何用调合的颜色或喷涂的颜色填充复杂对象。

第 10 章中的 Mini-Bézier 程序

Mini-Bézier 程序所用的代码结构与 Artisan 程序用来绘制 Bézier 曲线的结构完全相同。

我把这个简化版本包括进来，是为了让你能够很容易地明白我是怎样逐点绘制 Bézier 曲线的。预绘制曲线所使用的两个过程在采用递归方式绘制 Bézier 曲线的过程中可能并不需要。不过，在两种方法中，算法都是一样的，只是采用预绘制过程速度稍稍快一些。

第 11 章中的绘制圆角程序

在 APIdraw 程序中，可以使用 API 的 RoundedRect 函数绘制带圆角的矩形。如果试图用同样的调用来移动该对象，那么该对象就会在各种绘图模式之间来回切换，并发生闪烁。第 11 章告诉你如何用 Basic 的 Line 和 Circle 语句绘制同样形状，但不闪烁的圆角矩形。

第 12 章中的 Text Alignment 程序

Visual Basic 提供了简单的在窗体上显示文字的方法。当你想在用户图形软件包的演示部分处理各种文字时，Windows API 文字函数在功能上就会显得略有不足。如果不涉及文字的 Glyph 轮廓并设计自己的文字模块，就不会有更简便的方法对文字进行处理。但对这些模块代码的解释会使本书显得过于冗长。本章展示了发生鼠标按下事件时设置插入点的三种简单的文字对齐方法：左对齐，右对齐或文字居中。

第 13 章中的节点连接程序

这是所有绘图包的核心和灵魂。第 2 章的 Artisan 程序中没有包括该程序，这是因为在第 2 章必须有一个明确的目标，想要让节点和对象做些什么。设计 Artisan 程序是为了避免调用 API 函数而只使用标准的 Basic 语言。当涉及填充对象时，就达到了 Basic 语言绘图能力的极限。你必须求助于 Pset 或 Line 语句的循环来填充复杂对象（因为在 Windows 中 Basic 的 Paint 语句被闲置不用了）。这时候，你就必须一头扎进 Windows API 绘图库中。本程序所用的源代码虽然简单，但是仍需要大量的变量存储单元。当你加入可拆分的对象节点和 Bézier 曲线连接时，你就可以深入到代码中。本书的第 1 章介绍了在开始一个绘图程序之前必须对所用的变量和数组作出安排。

第 14 章中的 PrintAPI 和 Scaltest 程序

PrintAPI 是一个简单的小程序，该程序测试打印机的 Windows API 图形函数。它使用 API 调用在一张标准的信笺页上画方框、图及直线。你可以将图形传送至打印机，然后加进额外的代码调整设备（监视器或打印机）的坐标比例。第二个程序（Scaltest ）包含了若干种缩放某个窗口区域的方法（既用了 Basic 方法，又用了 API 方法）。你还应该运行一下附带磁盘中 16MISC 目录下的 APIScale 程序，看一看缩放窗口内选定对象的例子。

第 15 章中的 ArtAPI 程序

ArtAPI 是第 2 章中 Artisan 程序的改进版本。不同之处是该程序用 Windows API 的绘图函数取代了 Basic 语句。从该章的简介开始对程序作了全面的说明。在探讨 ArtAPI 程序之前，一定要读一下第 2 章的 Artisan 程序和第 13 章的节点连接程序。

附录 A

附录 A 包含了 Artisan 程序的全局变量清单，以及该程序中所用的文字和调色板窗体。

附录 B

附录 B 包含了与 Windows NT，Windows 95 有关的信息。

附录 C

附录 C 是对附带磁盘中 16MISC 目录下程序的简介。对这些附加程序代码的解释显示在每个程序的 Visual Basic 代码窗体中。这些程序展示了实现下列功能的例子：

- 怎样创建自定义的工具栏和按钮
- 怎样在 Visual Basic 2.0 和 3.0 中创建自定义光标
- 怎样创建标签风格的对话窗体
- 怎样在运行时确定窗体的位置和比例
- 怎样给其他程序增加菜单项
- 怎样使用少量的或不用资源显示位图图象
- 怎样使用 DPI 打印那些采用自定义比例尺寸的窗体或控制图象
- 怎样创建 Visual Basic 和 API 视窗
- 以及……

图形用户界面(GUI)

一个类似 Windows 的图形用户界面与 Basic 的图形语句相似，因为它采用隐含的代码创建图形对象。Windows 包含了一个 API 函数库，你可以利用这些函数完成超出标准 Basic 语句所能实现的功能。DOS 操作系统中的最后一个 Basic 版本有大约 200 个函数，而 Windows API 库则有 1000 多个。仅在图形库中，你就有若干种方法简单地绘出比直线和圆更复杂的图形，而且不用编写很长的例程就可以达到与 Basic 同样的效果。不过你必须对如何将这些直线和圆(或其他对象)放置在监视器屏幕上有所了解。在你充分理解这些以后，你就可以着手实现 Windows API 中的这些高级功能了，以后的章节中会讲述上述内容。

Basic 和 Windows

你在使用 Visual Basic 开发 Windows 程序方面有了一个良好的开端。到目前为止的最新版本有 400 多条不同的 Basic 语句。将这些和前面提到的 API 函数结合起来，你就可以创建一些强有力的应用程序。Visual Basic 在执行速度方面可能没什么特别，但在快速开发 Windows 程序上，它却得到了一致的首肯。

机器代码、解释器和编译器

有两种计算机语言：低级的和高级的。低级语言通常被称作机器代码或汇编语言。最基础的就是机器代码，它使用一些十六进制的指令直接与 CPU 对话。再向上一级是汇编语言，它使用一些简写(缩写)了的机器过程操作 CPU，这让人稍能容忍。

再向上一级就是高级语言。高级语言仅仅是一种称呼而已，它并不是指你正在使用的语言有多么高级。你不应该对使用这些语言有任何犹豫。Visual Basic 可以被称为高级语言，它被划入解释语言一类。与之相对的就是 C、Pascal 以及类似的语言，它们采用高级的编译技术进行处理。两者之间的差别在于像 Basic 这样的解释程序是将每一条语句翻译成隐含的机器代码之后，CPU 再执行这些代码。Visual Basic 需要有一个单独的文件来将子程序翻译成机器代码，这个文件就是 VBrun，内部翻译器代码就被称作 P-code 解释码。因此你的程序并不真是百分之百编译的 . EXE 文件。

编译语言(例如 C)被转换成汇编程序，生成最终的机器代码。虽然在操作上与 P-code 解释代码类似，但是编译的代码就执行速度而言远远胜过解释的代码。解释的代码读进每一条 Basic 命令语句，然后随着代码的输入每次执行一行。而编译的代码被编译成单个文件。

在开始之前应该知道哪些内容

第 1 章让你理解在使用简单的图形语句时屏幕坐标系统是怎样运作的，还将给你上一堂三角学复习课(让你回忆一下在学校中所学过的数学知识)。不过，此节之后不会有什么小测验了。

在每一段源代码清单之后都有一段对每行代码所作的简短解释。在这些摘要说明中，每一代码行的头几个单词都是以黑体给出，后面再跟一段解释。

你应该已经对 Visual Basic 的基本概念比较熟悉了，本书假设你超出了初学者的水平。不过，书中有些部分还是回顾了一下基本概念，以确保你理解这些基本知识，如果你已经对 Visual Basic 有了基本的理解，那就再好不过了。

程序

为了让所有的程序都运行正常，我付出了极大的努力。许多程序都是用标准的函数和子程序编写的。

测试这些代码使用了许多机器，包括以下这些：i386sx 16，386AMD40，i486DX33，486DCLcyrix 以及 Pentium 60。

鉴于这些程序的图形特性，应该有一台分辨率高于标准 VGA 的监视器。大部分图形程序都是针对 Super VGA 监视器创建的。本书中的全部程序窗体都是设计成在 800×600 分辨率的 SVGA 监视器上运行的。

附注： 所有程序的代码都包括在附带的磁盘中。另外还包括一些其他的程序。有关本书附带的磁盘所包含的内容的信息，请参见附录 C。

目 录

简 介 …… (1)

第 1 章 基本图形的绘制 …… (1)

1.1 处理圆 …… (3)
  1.1.1 选定绘图颜色 …… (6)
  1.1.2 绘制 ColorBar 调色板 …… (6)
  1.1.3 Cls (Command1_Click)按钮 …… (7)
  1.1.4 在窗体上单击鼠标器 …… (7)
  1.1.5 选定一个绘制选项 …… (8)
  1.1.6 计算圆的半径 …… (8)
  1.1.7 最后的成品 …… (9)
  1.1.8 计算圆弧,饼图及扇形的弧度值 …… (11)
  1.1.9 Line 和 Circel 语句简介 …… (11)
1.2 使用 Pset 语句 …… (13)
  1.2.1 选择一个画图函数按钮 …… (17)
  1.2.2 在窗体上单击鼠标 …… (18)
  1.2.3 SineCosine 过程 …… (19)
  1.2.4 Box_Pset 过程 …… (20)
  1.2.5 Bresenham 方法 …… (21)
  1.2.6 Circle_Pset 过程 …… (23)
  1.2.7 Circle_sin_cos 过程 …… (24)
  1.2.8 Circle_Sqr 过程 …… (24)
  1.2.9 Dash_Pset 过程 …… (25)
  1.2.10 Fill_Line 过程 …… (25)
  1.2.11 Fill_Pset 过程 …… (26)
  1.2.12 Grid_Pset 过程 …… (27)
  1.2.13 Line_Pset 过程 …… (28)
  1.2.14 Visual Basic 的 Point 语句 …… (28)
  1.2.15 Pset 语句小结 …… (29)
1.3 移动图形对象 …… (30)
  1.3.1 单击 Cls 按钮 …… (33)
  1.3.2 选定一个移动按钮 …… (33)
  1.3.3 Form_MouseDown 事件 …… (34)
  1.3.4 Form_MouseMove 事件 …… (36)
  1.3.5 Form_MouseUp 事件 …… (38)

1.3.6 Movetest 程序小结 …… (39)

1.4 本章小结 …… (39)

第2章 Artisan 程序 …… (40)

2.1 输入或输出过滤器 …… (40)

2.2 页面设置、标尺和边界打印尺寸 …… (41)

2.3 所见即所得(WYSIWYG) …… (41)

2.4 屏幕、打印机和分辨率 …… (41)

2.5 程序间的协调 …… (42)

2.6 在启动时自动重画(AutoRedraw) …… (42)

2.7 学习过程 …… (43)

2.8 程序速度和图片框 …… (43)

2.9 工具按钮图片框 …… (44)

2.10 颜色栏调色板 …… (45)

2.11 滚动条控制对象 …… (46)

2.12 Status Area(状态区域)框 …… (47)

2.13 菜单区域 …… (47)

2.13.1 File 菜单 …… (50)

2.13.2 Edit 菜单 …… (50)

2.13.3 Arrange 菜单 …… (50)

2.13.4 Color 菜单 …… (50)

2.13.5 Color Palette 子菜单 …… (50)

2.14 鼠标器事件 …… (51)

2.15 定义变量、数组和结构 …… (56)

2.15.1 数据类型 …… (57)

2.15.2 用户定义的数据类型变量 …… (58)

2.15.3 为 Artisan 程序声明变量 …… (59)

2.15.4 启动时默认的窗体装载值 …… (66)

2.16 确定控制对象在 Artisan 窗体中的位置 …… (69)

2.17 从工具箱选定一个绘图工具 …… (71)

2.18 在窗体第一次启动时就按下某个按钮 …… (73)

2.19 单击窗口中的工具按钮 …… (73)

2.20 在桌面上拖曳或移动鼠标 …… (74)

2.21 完成绘图 …… (75)

2.22 直线到 Bezier 曲线 …… (78)

2.23 在桌面上输入字符 …… (79)

2.24 在桌面上重画文字 …… (79)

2.25 BoundingBox_MouseDown 过程 …… (80)

2.26 在桌面上执行实际的绘图操作 …… (81)

2.27 给新对象指定一个引用号 …… (82)

2.28 本章小结 …… (85)

第 3 章 Bézier 曲线 …………………………………………………………… (86)

- 3.1 绘制 Bézier 曲线点 …………………………………………………… (90)
- 3.2 deCasteljau 分划公式 ………………………………………………… (93)
- 3.3 通过图柄拉伸曲线 …………………………………………………… (94)
- 3.4 重画 Bézier 控制线(lever) …………………………………………… (95)
- 3.5 存储曲线的新坐标点 ………………………………………………… (96)
- 3.6 曲线的第二个 Bézier 图柄 …………………………………………… (97)
- 3.7 将一条直线转换成一条曲线 ………………………………………… (99)
- 3.8 捡取一条曲线 ………………………………………………………… (101)
- 3.9 移动前擦除曲线图象 ………………………………………………… (103)
- 3.10 为曲线建立一个边界方框 …………………………………………… (104)
- 3.11 曲线上控制线(lever lines)的视觉状态 ……………………………… (105)
- 3.12 拉伸 Bézier 曲线 ……………………………………………………… (108)
- 3.13 本章小结 ……………………………………………………………… (111)

第 4 章 Artsian 程序— 过程 C 到 N …………………………………… (112)

- 4.1 指定对象的边框和填充颜色 ………………………………………… (112)
- 4.2 ARTISAN.INI 文件 …………………………………………………… (114)
- 4.3 用选定方框搜索一个对象 …………………………………………… (114)
- 4.4 在对象上单击鼠标 …………………………………………………… (116)
- 4.5 绘制页标线 …………………………………………………………… (119)
- 4.6 定位对象周围的拉伸图柄 …………………………………………… (120)
- 4.7 逆置翻转对象的图柄位置 …………………………………………… (123)
- 4.8 准备一个待拉伸的对象 ……………………………………………… (125)
- 4.9 拉伸一个对象 ………………………………………………………… (128)
- 4.10 在桌面上滚动图形 …………………………………………………… (131)
- 4.11 改变对象边框的宽度 ………………………………………………… (133)
- 4.12 Artisan 程序的主菜单选项 …………………………………………… (135)
- 4.13 从桌面上删除一个单独的对象 ……………………………………… (135)
- 4.14 为 File 菜单选项做准备 ……………………………………………… (136)
- 4.15 删除桌面上的全部对象 ……………………………………………… (137)
- 4.16 Color 菜单项选项 …………………………………………………… (138)
- 4.17 改变颜色调色板中的颜色 …………………………………………… (138)
- 4.18 计算鼠标移动 ………………………………………………………… (139)
- 4.19 在桌面上移动对象 …………………………………………………… (140)
- 4.20 确定对象周围的节点的位置 ………………………………………… (141)
- 4.21 从桌面上移去节点和图柄 …………………………………………… (143)
- 4.22 擦去一个对象的填充颜色 …………………………………………… (144)
- 4.23 本章小结 ……………………………………………………………… (144)

第 5 章 Artisan 程序——过程 N 至 Z ……………………………………………… （145）

5.1 通过节点移动一个对象 ……………………………………………… （145）
5.2 移动直线对象的节点 ……………………………………………… （147）
5.3 移动节点后更新直线对象 ……………………………………………… （148）
5.4 使用弹出式绘图工具填充一个对象 ……………………………………………… （149）
5.5 从 .INI 文件中抽取颜色值 ……………………………………………… （150）
5.6 创建正圆 ……………………………………………… （151）
5.7 用 Visual Basic 在桌面上放置文字 ……………………………………………… （152）
5.8 重新设定节点和图柄的内部坐标比例 ……………………………………………… （154）
5.9 标尺的鼠标器事件 ……………………………………………… （155）
5.10 绘制每个标尺的滑线 ……………………………………………… （156）
5.11 在每个标尺内绘制英寸刻度 ……………………………………………… （157）
5.12 将值写进 ARTISAN.INI 文件 ……………………………………………… （159）
5.13 将某个对象放到所有其他对象的前面或后面 ……………………………………………… （160）
5.14 重画桌面上的所有对象 ……………………………………………… （163）
5.15 重画隐含页面 ……………………………………………… （166）
5.16 向打印机传送图象 ……………………………………………… （168）
5.17 使用缩放工具 ……………………………………………… （170）
5.18 重新设定桌面视口的比例 ……………………………………………… （173）
5.19 本章小结 ……………………………………………… （174）

第 6 章 Animotor 程序 ……………………………………………… （175）

6.1 启动时的默认值 ……………………………………………… （182）
6.2 设置 3-D 动画图片框的比例 ……………………………………………… （183）
6.3 开始 3-D 动画进程 ……………………………………………… （184）
6.4 3-D 动画的计算 ……………………………………………… （185）
6.5 使用箭头来移动物体 ……………………………………………… （187）
6.6 动画的开关控制 ……………………………………………… （190）
6.7 动画程序中的网格控制对象 ……………………………………………… （191）
6.8 从不同角度看一个 3-D 物体 ……………………………………………… （192）
6.9 打开或存储一个数据文件 ……………………………………………… （193）
6.10 存储数据文件 ……………………………………………… （194）
6.11 打开并显示数据 ……………………………………………… （195）
6.12 文件中数据的更新 ……………………………………………… （196）
6.13 通过滚动条移动 3-D 物体 ……………………………………………… （196）
6.14 输入新的数据 ……………………………………………… （197）
6.15 本章小结 ……………………………………………… （198）

第 7 章 多重查找和替换 ……………………………………………… （200）

7.1 本程序中用到的变量 ……………………………………………… （205）

7.2 在启动时为控制对象定位 …… (205)
7.3 给代码列表加入条目 …… (206)
7.4 在列表框中选择条目 …… (207)
7.5 从列表框中删去条目 …… (208)
7.6 用于编辑文本的工具 …… (209)
7.7 编辑列表框中条目 …… (210)
7.8 怎样剪切、拷贝及粘贴文本 …… (211)
7.9 打开或保存一个文本文件 …… (211)
7.10 打开一个文本文件 …… (214)
7.11 查找并替换条目 …… (216)
7.12 保存文本文件 …… (218)
7.13 文本间隔 …… (219)
7.14 本章小结 …… (223)

第 8 章 API 绘图程序 …… (225)

8.1 APIdraw 程序的声明部分 …… (227)
8.2 单击一个 API 函数按钮 …… (230)
8.3 窗体的鼠标事件 …… (232)
8.4 用 FloodFill 函数绘图 …… (235)
8.5 API 绘图函数 …… (235)
8.6 椭圆函数(Ellipse) …… (236)
8.7 读象素函数(Get_Pixel) …… (236)
8.8 画线函数(Line_to) …… (236)
8.9 API 饼图函数(Pie) …… (237)
8.10 多边形函数(Polygons) …… (238)
8.11 折线函数(Polyline) …… (239)
8.12 Pt_In_Rect 函数 …… (239)
8.13 矩形函数(Rec_tangle) …… (240)
8.14 圆角矩形函数(Round_Rect) …… (241)
8.15 **设置象素函数(Set_Pixel)** …… (241)
8.16 本章小结 …… (242)

第 9 章 填色与调色 …… (243)

9.1 控制颜色的结构体 …… (244)
9.2 Palette 程序的声明 …… (247)
9.3 Form_Resize 过程 …… (247)
9.4 两个主调色板 …… (248)
9.5 从调色板中选择一颜色 …… (249)
9.6 使 Fountain Fill 控制对象生效 …… (251)
9.7 API 调色板 …… (252)
9.8 显示颜色值的范围 …… (252)

9.9 显示一个纯色的调色板 …… (253)
9.10 创建一个自定义调色板(Custom Palette) …… (254)
9.11 PALETTE 和 PALETTEENTRY …… (255)
9.12 在调色板中使用抖色 …… (255)
9.13 绘制调色板上的每一颜色 …… (256)
9.14 Fountain Fill 按钮 …… (257)
9.15 使用 RGB 颜色值来建立一个调色板 …… (257)
9.16 通过滚动条来改变颜色 …… (259)
9.17 对一个复杂物体进行喷涂填色 …… (259)
9.18 本章小结 …… (260)

第 10 章 Bézier 曲线程序 …… (262)

10.1 16 位 Bézier 曲线程序使用的变量名 …… (264)
10.2 计算曲线上的点 …… (265)
10.3 绘出曲线上的点 …… (266)
10.4 在窗体内画一条线 …… (269)
10.5 移动前清除旧的线条 …… (271)
10.6 画一条新曲线前重置各初始值 …… (272)
10.7 Bézier 图柄 1 …… (273)
10.8 重绘图柄 1 的控制线 …… (273)
10.9 重设图柄 1 的大小 …… (274)
10.10 Bézier 图柄 2 …… (274)
10.11 Handle 2 的控制线 …… (275)
10.12 重设图柄 2 的大小 …… (275)
10.13 通过节点移动一条曲线 …… (275)
10.14 重设节点 1 …… (277)
10.15 通过 Node 2 移动曲线 …… (277)
10.16 将一条直线转换为一条曲线 …… (278)
10.17 本章小结 …… (279)

第 11 章 绘制圆角程序 …… (280)

11.1 如何使用本程序 …… (280)
11.2 声明及初始值 …… (282)
11.3 窗体的鼠标事件 …… (284)
11.4 通过 Basic 过程画一个圆角矩形 …… (285)
11.5 使用 Step 语句画圆角 …… (287)
11.6 选择一个绘制选项 …… (288)
11.7 用 Stretcher 过程绘制圆角矩形 …… (288)
11.8 使用滚动条控制对象 …… (289)
11.9 本章小结 …… (290)

第 12 章 文本对齐程序 ………………………………………… **(291)**

12.1 行间距(Leading) ………………………………………… (291)
12.2 字距(Width) ………………………………………… (292)
12.3 TxtAlign 程序 ………………………………………… (293)
12.4 在 Visual Basic 中手工放置文本 ………………………………………… (295)
12.5 TxtAlign 程序总结 ………………………………………… (297)
12.6 使用 APItext 程序 ………………………………………… (297)
12.7 APItext 程序的声明与结构 ………………………………………… (300)
12.8 默认的初始值 ………………………………………… (301)
12.9 在 API Text 中添加回车换行符 ………………………………………… (303)
12.10 向 TextOut 函数添加属性 ………………………………………… (306)
12.11 使用 SoaleLeft 属性放置文本 ………………………………………… (307)
12.12 显示 FontLog 属性 ………………………………………… (308)
12.13 使用 API 文本对齐函数 ………………………………………… (309)
12.14 显示一种字体的文本特征值(Metrics) ………………………………………… (309)
12.15 APItext 程序小结 ………………………………………… (311)
12.16 DrawText 程序 ………………………………………… (311)
12.17 变量名及启动时的初始化值 ………………………………………… (314)
12.18 用 DrawText 函数绘制文本 ………………………………………… (315)
12.19 API 文本绘制和文本高度 ………………………………………… (317)
12.20 DrawText 程序小结 ………………………………………… (318)
12.21 本章小结 ………………………………………… (319)

第 13 章 节点连接程序 ………………………………………… **(320)**

13.1 程序是如何工作的 ………………………………………… (320)
13.2 变量名和初始值 ………………………………………… (321)
13.3 鼠标事件 ………………………………………… (324)
13.4 绘制折线段 ………………………………………… (326)
13.5 复杂对象的填色 ………………………………………… (327)
13.6 使用节点编辑多边形 ………………………………………… (329)
13.7 移动连接的线段 ………………………………………… (331)
13.8 增加更多的图片框节点 ………………………………………… (332)
13.9 装载对象的所有节点 ………………………………………… (333)
13.10 绘制最终的多边形 ………………………………………… (334)
13.11 节点连接程序中的 API 函数 ………………………………………… (335)
13.12 本章小结 ………………………………………… (335)

第 14 章 定标与打印程序 ………………………………………… **(336)**

14.1 变量名和 API 函数 ………………………………………… (338)
14.2 在图片框上绘制轮廓 ………………………………………… (339)

14.3 绘制对象 …… (340)
14.4 打印机上页面的人工定标 …… (340)
14.5 API 打印 …… (341)
14.6 PrintAPI 程序小结 …… (342)
14.7 定标测试程序 …… (343)
14.8 声明与初始值 …… (346)
14.9 图片框鼠标事件 …… (347)
14.10 初始化定标值对齐 …… (349)
14.11 Zoom API 菜单 …… (350)
14.12 Zoom Basic 菜单 …… (351)
14.13 重定标图片框控制对象 …… (352)
14.14 本章小结 …… (353)

第 15 章 ArtAPI 程序 …… (354)

15.1 图柄与节点 …… (354)
15.2 坐标系统与缩放工具 …… (354)
15.3 桌面及打印机 API 图形 …… (354)
15.4 ArtAPI 程序中的图形 API 函数 …… (355)
15.5 ArtAPI 程序中的主要变动 …… (355)
15.6 新的图片控制对象 …… (355)
15.7 变为过程的原有控制 …… (356)
15.8 其他改动 …… (356)
15.9 新的多边形过程 …… (356)
15.10 To_Front_Back 例程 …… (358)
15.11 API 定标系统和 WinPrint(窗口打印) …… (358)
15.12 测试(Test)按钮 …… (358)
15.13 缩放工具 …… (358)
15.14 ArtAPI 窗体中新的声明 …… (364)
15.15 将折线转换为曲线 …… (365)
15.16 Artwork 鼠标事件 …… (367)
15.17 在桌面上重绘多边形 …… (368)
15.18 Pnode 图象控制的 Tag 属性 …… (370)
15.19 更新曲线的坐标 …… (375)
15.20 移动对象 …… (380)
15.21 API 绘图函数 …… (381)
15.22 寻找对象的轮廓 …… (383)
15.23 为多边形计算边界框 …… (384)
15.24 打开与保存多边形文件 …… (385)
15.25 移动多边形对象 …… (386)
15.26 多边形的节点(Pnode) …… (388)
15.27 编辑多边形中的曲线 …… (395)

15.28 绘制多边形的每一条曲线 …… (399)
15.29 封闭一个多边形 …… (401)
15.30 绘制折线形状 …… (403)
15.31 编辑折线形状 …… (404)
15.32 载入并重新定位多边形上的 Pnode …… (406)
15.33 重新载入多边形上的 Pnode …… (407)
15.34 在编辑多边形线段之后 …… (409)
15.35 调整多边形大小和拉伸多边形 …… (409)
15.36 使用测试按钮 …… (411)
15.37 将对象移到前面或后面 …… (412)
15.38 使用轮廓图象选择一对象 …… (413)
15.39 在桌面上绘制所有对象 …… (415)
15.40 打印所有对象 …… (419)
15.41 本章小结 …… (422)

附录 A Artisan. BAS 文件 …… (424)

A.1 ART3_7.BAS 文件 …… (424)
A.2 ART3_7.BAS 文件中的全局声明 …… (424)
A.3 给调色板赋以颜色 …… (431)
A.4 给对象赋以颜色 …… (432)
A.5 Artisan1 Text 窗体 …… (432)
A.6 Text 窗体的总结 …… (438)
A.7 Artisan NodeEdit 窗体 …… (439)
A.8 Artisan ColorPanlette(彩色调色板)窗体 …… (440)
A.9 调色板窗体使用的值 …… (440)
A.10 Palette 窗体的总结 …… (445)

附录 B Windows NT.Windows 95 及未来的 Windows 版本 …… (446)

B.1 GDI32 的路径函数 …… (446)
B.2 GDI32 StrokePath …… (447)
B.3 GDI32 StrokeAndFillPath …… (447)
B.4 GDI32 Fill Path …… (447)
B.5 GDI32 GetPath …… (447)
B.6 GDI32 Select ClipPath …… (447)
B.7 GDI32 PathToRegion …… (447)
B.8 GDI32 WidenPath …… (448)
B.9 GDI32 的绘图函数 …… (448)
B.10 GDI32 AngleArc …… (448)
B.11 GDI32 ArcTo …… (448)
B.12 GDI32 SetArcDirection …… (448)
B.13 GDI32 CloseFigure …… (448)

B.14 GDI32 PolyBézier …………………………………… (448)
B.15 GDI32 PolyBézierTo …………………………………… (449)
B.16 GDI32 PolyDraw …………………………………… (449)
B.17 GDI32 EnumEnhMetaFile …………………………………… (449)
B.18 GDI32 MaskBit …………………………………… (449)
B.19 GDI32 PlgBit …………………………………… (449)
…………………………………… (449)

附录 C 书中所附磁盘 …………………………………… (451)

C.1 盘上文件的安装 …………………………………… (451)

第 1 章　基本图形的绘制

本章的 Begin 程序(图 1. 1)是为初学者设计的。Begin 程序可以帮助你理解基本的图形语句。当你掌握了在监视器屏幕上绘制简单的线或圆所必需的代码结构后，你就可以更好地理解 Basic 的内部运行机制和 Windows 的图形环境。对于理解图形的内在结构来说，Line 语句和 Circle 语句是最基本的。例如，Windows 中的窗口就是由线和圆组成的，这些线和圆构成了矩形或椭圆形的形状。

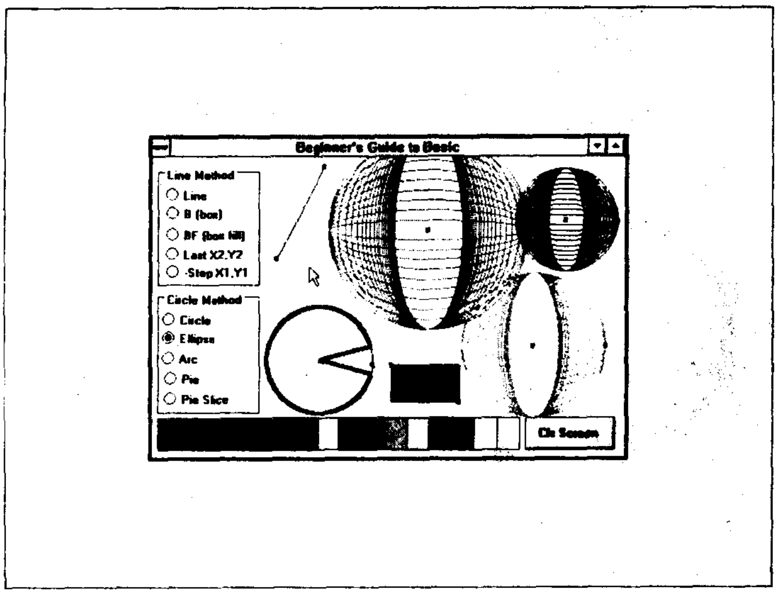

Beginner's Guide to Basic
Line Method
○ Line
○ B (box)
○ BF (box fill)
○ Last X2,Y2
○ -Step X1,Y1
Circle Method
○ Circle
● Ellipse
○ Arc
○ Pie
○ Pie Slice
Clr Screen

图 1. 1　Begin 程序

Line 方法

Line(X1,Y1)-(X2,Y2)
Line-Step(X2,Y2)
Line-(X2,Y2)
Line(X1,Y1)-Step(X2,Y2)

Circle 方法

R=半径,C=颜色
S=起始,E=终止,A=纵横比
Circle(X,Y),R
Circle(X,Y),R,C,S,E,A

选项:

C=颜色,B=方框的边框
BF=用颜色填充的方框

Line(X1,Y1)-(X2,Y2),Color
Line(X1,Y1)-(X2,Y2),C,B
Line(X1,Y1)-(X2,Y2),C,BF

绘制基本对象的例子参见图 1.2。

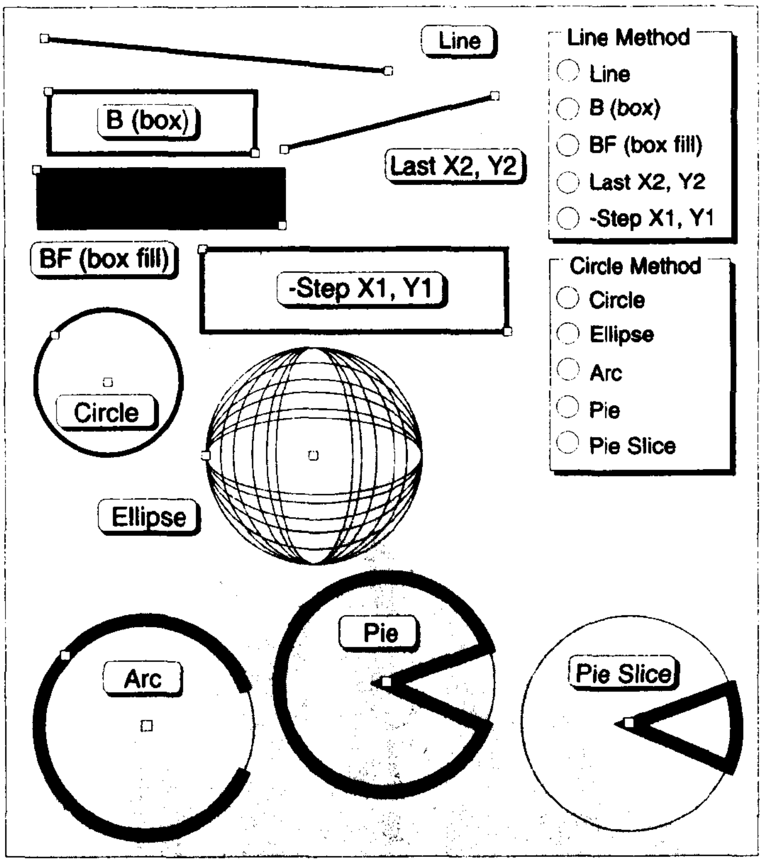

Line
Line Method
○ Line
○ B (box)
○ BF (box fill)
○ Last X2, Y2
○ -Step X1, Y1
B (box)
Last X2, Y2
BF (box fill)
-Step X1, Y1
Circle Method
○ Circle
○ Ellipse
○ Arc
○ Pie
○ Pie Slice
Circle
Ellipse
Pie
Arc
Pie Slice

图 1.2 绘制基本的线、边框方框、填充的方框、圆、椭圆、圆弧、饼图和扇形

在 Visual Basic 中，Control.Left(左)和 Top(上)属性确定了该控制对象的左上角在窗体网格坐标系中的位置，该坐标反映了窗体的当前 ScaleMode(比例模式)。

遗憾的是，Visual Basic 的控制对象没有直接指定其右侧或下方位置的属性。若想要确定一个控制对象的右下角位置，就必须使用它的 Width(宽)和 Height(高)属性。而且，如果你试图将一个控制对象置于某个窗体的中央，你就很可能希望窗体或控制对象还有一个置中属性。然而 Basic 的 Line 语句则是按屏幕坐标(X1,Y1)－(X2,Y2)设置的。

假如你用 Line 语句绘制了一个方框形状，并且是从靠近窗体左上角的地方开始，到靠近窗体右下角的地方终止，那么就可以说该方框是按左/上和右/下坐标设定的。实际情况下，X1、Y1 坐标指示的是方框的起始点，而 X2，Y2 坐标指示的是方框的终止点。一个方框有四个角，所以有四种可能的起始点、终止点方案：

1. (左，上)－(右，下)
2. (右，上)－(左，下)
3. (右，下)－(左，上)
4. (左，下)－(右，上)

使用 X，Y 网格坐标模式，一条线或一个方框可以起始于屏幕上的任何位置，也可以终止于屏幕上的任何位置。X，Y 坐标系是基于起始原点 0,0 的，从屏幕的左上角开始。正 X 值

表示从原点开始向右递增，而正 Y 值表示向屏幕的底边递增。可以这样来想象，X 代表网格的列，而 Y 代表行。你可以从左向右来看这个不可见的网格。不过，应该指出的是，Visual Basic 和 API 的缩放比例法可以将左上角的 0，0 原点置于任何位置，而且可以切换 X 轴和 Y 轴的方向。如果将位于 0，0 原点下方点的 Y 坐标设成负数（笛卡尔坐标），这在某些情况下是很有用的。

图 1.3 给出了三条 Line 语句的左-上和右-下点的值。你并不一定非要用变量名（X1，Y1）；（X2，Y2）来作这些对象的位置的临时存储单元。本书的大多数程序都是用（W，N）；（E，S）（西，北，东，南），这样可以更好地把握对象的顶点的位置。

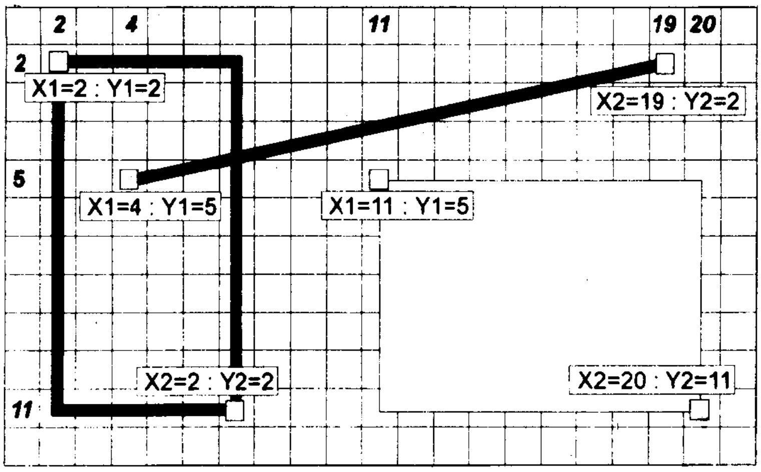


图 1.3　三条 Line 语句的左-上和右-下值

图 1.3 中网格顶部从左至右的粗黑体数字指示出各个对象的当前 X 坐标，而网格左边从顶至底的粗黑体数字指示的是这些对象的 Y 坐标。

1.1　处理圆

一个圆的大小是由它的半径长度来决定的。图 1.4 中的两个黑点代表半径的（X1，Y1）；（X2，Y2）坐标。X1，Y1 点代表圆心，而 X2，Y2 点代表圆周上的某个位置。从 X1，Y1 点到 X2，Y2 点画一条线，这条线的长度就是该圆的半径。

当半径线与坐标轴有一定的夹角时，就必须利用一个公式来算出半径线的长度。计算圆半径的公式叫作勾股定理。斜向的半径线叫作弦（斜边）。勾股定理是这样描述的：斜边的平方等于两侧边的平方和。按外行的话来说，就是必须首先求出 X2 到 X1 的水平长度和 Y2 到 Y1 的垂直长度，将这两个长度值自乘之后加在一起，然后求出总长度值的平方根。本章后面“计算圆的半径”一节将解释这一原理。

Begin 程序的窗体及控制对象的属性在表 1.1 中列出。

表 1.1　Begin 窗体的属性

Object：Form

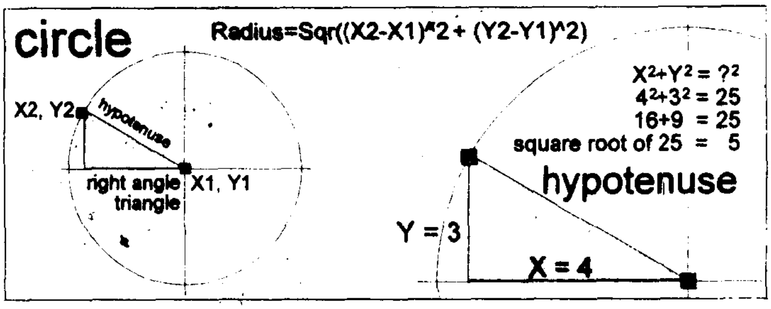

circle
Radius=Sqr((X2-X1)^2 + (Y2-Y1)^2)
X2, Y2
hypotenuse
right angle triangle
X1, Y1
X²+Y² = ?²
4²+3² = 25
16+9 = 25
square root of 25 = 5
hypotenuse
Y = 3
X = 4

图 1.4 根据勾股定理求出圆的半径

Object Name:Begin
Height = 5040 Left = 210
ScaleMode = 3' Pixel Top = 1125
Width = 7455

Object:Frame
Object Name:Frame2
Caption = "Circle Method" Height = 1905
Left = 90 Top = 2070
Width = 1590

Object:OptionButton
Object Name:Option1
Caption = "Pie Slice" Height = 285
Index = 9 Left = 90
Top = 1530 Width = 1275

Object:OptionButton
Object Name:Option1
Caption = "Pie" Height = 285
Index = 8 Left = 90
Top = 1215 Width = 1275

Object:OptionButton
Object Name:Option1
Caption = "Arc" Height = 285
Index = 7 Left = 90
Top = 900 Width = 1275

Object:OptionButton
Object Name:Option1
Caption = "Ellipse" Height = 285
Index = 6 Left = 90
Top = 585 Width = 1275

Object:OptionButton
Object Name:Option1
Caption = "Circle" Height = 285
Index = 5 Left = 90
Top = 270 Width = 1275

Object:CommandButton
Object Name:Command1
Caption = "Cls Screen" Height = 510
Index = 5805 Top = 4050

Width = 1410

Object:PictureBox
Object Name:ColorBar
Height = 510　　Left = 90
ScaleMode = 0' User　　Top = 4050
Width = 5640

Object:Frame
Object Name:Frame1
Caption = "Line Method"　　Height = 1860
Left = 90　　Top = 135
Width = 1590

Object:OptionButton
Object Name:Option1
Caption = "-Step X1,Y1"　　Height = 285
Index = 4　　Left = 90
Top = 1530　　Width = 1275

Object:OptionButton
Object Name:Option1
Caption = "Last X2,Y2"　　Height = 285
Index = 3　　Left = 90
Top = 1215　　Width = 1275

Object:OptionButton
Object Name:Option1
Caption = "BF(box fill)"　　Height = 285
Index = 2　　Left = 90
Top = 900　　Width = 1275

Object:OptionButton
Object Name:Option1
Caption = "B(box)"　　Height = 285
Index = 1　　Left = 90
Top = 585　　Width = 1230

Object:OptionButton
Object Name:Option1
Caption = "Line"　　Height = 285
Index = 0　　Left = 90
Top = 270　　Width = 1185

清单1.1是Begin程序运行期间所用全部变量名字的声明。如果你是一个初学者或对变量的概念还不太熟悉，可以参见第2章“Artisan程序”开始部分的一段简短的说明。

清单1.1　Begin窗体的声明段

```
Declarations
Dim CurrentOption As Integer
Dim X1, Y1, X2, Y2 As Integer
Dim Radius
Dim Lcolor As Long
Dim NextPoint As Integer
Const PI = 3.141593    'the value of PI
```

附注： 本书是这样组织的：每一段程序或过程源代码后面紧跟着逐行的解释说明。每段说明开始处的粗黑体字代表实际的源代码行最开始的关键字。

CurrentOption： 保存当前选定的选项按钮索引号。本例中共有 10 个选项按钮，其索引号从 0 到 9。这样，要搞清楚用户选定了哪个选项按钮就很容易了。Form_MouseUp 事件中的 Select Case 语句将每一个选项按钮索引对应一个要绘制的对象。

X1，Y1，X2，Y2： 这几个变量保存执行 MouseDown 事件时屏幕上两点的位置。

Radius： 该变量保存由勾股定理计算出的圆半径。该值目前不必保存，因为该值只在一个过程事件中用到。

Lcolor： 存储用来显示线或圆的外框及填充颜色的 16 进制颜色值。保存 16 进制颜色值时，应该使用 Long（长整型）变量数据类型。有关变量类型和值的解释请参见第 2 章的开始部分。

NextPoint： 在代码内鼠标按下事件之间将该变量切换打开或关闭。

附注： 所有事件和过程的代码都紧接在声明段之后。事件或过程的源代码清单按字母顺序排放。如果某段源代码非常复杂，则在文字段内会有更详细的解释。

1.1.1 选定绘图颜色

清单 1.2 ColorBar_MouseDown 事件

```
Sub ColorBar_MouseDown(Button%,Shift%,X!,Y!)
    Lcolor=ColorBar.Point(X,Y)
End Sub
```

Lcolor＝ColorBar.Point(X,Y)： Point 语句返回调色板中选定颜色的 16 进制颜色值。此语句只对纯颜色起作用，因为抖动处理过的颜色是由几种颜色组合成的。Point 语句返回一个单色象素的值。对于抖动处理过的颜色来说，某一单个象素可能并不是所希望的颜色。彩色调色板中所用的 16 种颜色是标准的 QBColor——标准的 Basic 语言颜色值（0 至 15）。第 2 章中的 Artisan 程序展示了如何选定更大的颜色值。

1.1.2 绘制 ColorBar 调色板

彩色调色板是一个图片框。它的内部按 16 个单位宽、1 个单位高划分为 16 个相等的方块，每个方块设置一种颜色。清单 1.3 中列出的 ColorBar_Paint 可绘出该彩色调色板，并划分出该彩色调色板的分隔线。

清单 1.3 ColorBar_Paint 事件

```
Sub ColorBar_Paint()
    ColorBar.Scale (0,0)-(16,1)
        For I=0 To 15
            ColorBar.Line (I,0)-(I+1,1),QBColor(I),BF
            If I Then ColorBar.Line(I,0)-(I,1)
        Next I
```

```
End Sub
```

For I=0 To 15：　对于从 0 至 15 的每一个 I 值，执行下列操作：

ColorBar. Line：用从 0 至 15 的 QBColor 填充 ColorBar 的全部 16 个方块。

If I Then：在每个颜色块之间画一条细黑的分隔线。

1.1.3　Cls（Command1_Click）按钮

Command1_Click 按钮可擦掉窗体上的所有图象，以便绘制新的图形。清单 1.4 列出了 Command1_Click 的代码，它完成擦除窗体上所有图象的操作。

清单 1.4　Command1_Click 子程序

```
Sub Command1_Click()
    Cls
End Sub
```

清单 1.5 列出了 Form_Load 事件的代码，它可在启动时设置变量的默认值。

清单 1.5　Form_Load 子程序

```
Sub Form_Load()
    NextPoint=False
    Lcolor=QBColor(5)
End Sub
```

1.1.4　在窗体上单击鼠标器

Form_MouseDown 事件设置所要绘制对象的起始位置。它用一个 Select/Case 块（Case 设置从 0 至 9）设置所绘对象的起始坐标。

附注：　*Begin，Pset 和 MoveTest 程序都采用 Select/Case 块选定所需的代码执行。在某些控制事件的处理过程中，Select/Case 语句并不是必须使用的。之所以将其加入是为了便于读者在需要时加入自己的代码。在某些 Select/Case 块中，类似的代码多次重复出现，而不是将其编写成一个过程。因此，你可以剪切并粘贴这些代码块以供自己使用。*

Form_MouseDown 子程序请参阅清单 1.6。

清单 1.6 中的 Form_MouseDown 设置了绘制一个对象所需的初始值。

清单 1.6　Form_MouseDown 子程序

```
Sub Form_MouseDown (Button% , Shift%, X As Single, Y As Single)
If NextPoint = False Then
    Select Case CurrentOption
        Case 0 To 9 'Line, Box, Box Filled
        X1 = X: Y1 = Y
        DrawWidth = 4: PSet (X1, Y1): DrawWidth = 1
    End Select
End If
End Sub
```

If NextPoint＝False Then：为了能在第一次鼠标键按下事件发生时设置 X1，Y1 屏幕位置，必须抑制第二次鼠标键按下事件。然后在鼠标键第二次抬起时设置终点 X2，Y2 屏幕位置。每当鼠标键抬起事件触发时，程序中使用 Not 操作符来回切换 True 和 False（开/关）值（见清单 1.8 中 Form_MouseUp 事件的第二至最后一行）。

Select Case CurrentOptions：此条 Select/Case 语句是不必要的，但是如果想要加入其他的可能使用该代码结构的选项时，就很方便了。

X1＝X：Y1＝Y：设置直线段在屏幕上的第一点，或者某个圆的中心位置。

DrawWidth＝4：Pset（X1，Y1）：DrawWidth＝1：将画线宽度设成 4 个象素宽，然后在鼠标键单击的地方画一个点，最后再将绘图宽度设成默认的 1 个象素宽。

1.1.5　选定一个绘制选项

当窗体上发生一次鼠标器单击事件时，如果某个选项按钮被选定，将激活相应的绘图例程。该绘图例程是利用 Select/Case 表达式通过按钮的索引号选定的。窗体鼠标器单击事件中所用的 Select/Case 测试表达式就是 CurrentOption 变量的值。清单 1.7 中的 Option1_Click 事件将 CurrentOption 变量设置成指定给该选项按钮的索引号。

清单 1.7　Option1_Click 事件

```
Sub Option1_Click(Index As Integer)
    CurrentOption = Index
End Sub
```

当你选定某个选项按钮时，变量 CurrentOption 的值就变成该选项按钮的索引号。

1.1.6　计算圆的半径

可以通过求直角三角形最长边的长度的方法来计算出一个圆的半径，这条最长的边为斜边（弦）。

如果将斜边的起点置于圆的中心上，那么，它的终点就应该落在该圆的圆周上。此斜边的长度就是该圆的半径。直角三角形的斜边就是与直角相对的边。

勾股定理

斜边的平方等于另两边的平方和。假设直角三角形的两条较短的边分别是 3 和 4 个单位长，那么，未知的斜边长就应该为 5。

（3×3）＋（4×4）＝？ × ？

简化为：

9＋16＝？ × ？

很显然 9＋16＝25，所以问题简化为“什么值自乘＝25？”。答案是 5，也就是 25 的平方

根。

本例中，直角三角形最长的边长等于5。这条边就是斜边，它等于圆的半径。

1.1.7 最后的产品

在设定了第二个屏幕点(X2,Y2)后，鼠标键抬起事件就触发了对象的绘制过程。即使你选择的是完全不同的画线选项按钮，画出的线或框看上去也没什么区别。实际上，绘制的确实是同样的线对象，只不过使用了不同的画线方法而已。

Line语句有若干可选的参数，加上这些参数就可以改变画线或画框的方法。关键字Step就是其中的一个例子，这个参数可以用来绘制一条相对于控制对象CurrentX和CurrentY属性值的线，或绘制一条相对某点X和Y坐标的线。

很多人都使用Step关键字来绘制连续的复杂折线。不过有时用多条Line语句获得理想的线型可能更为方便。在附带软盘中16MISC目录下Tabs程序中可以找到一个绘制混合线的很好例子。

清单1.8　Form_MouseUp事件，该事件在窗体上绘制最终的对象

```
Sub Form_MouseUp (Button%, Shift%, X As Single, Y As Single)
X2 = X: Y2 = Y
If NextPoint = True Then
    Select Case CurrentOption
      Case 0 'Line
        Line (X1, Y1)-(X2, Y2), Lcolor
      Case 1 'Box
        Line (X1, Y1)-(X2, Y2), Lcolor, B
      Case 2 'Box Filled
        Line (X1, Y1)-(X2, Y2), Lcolor, BF
      Case 3 'From Last X2, Y2
        Line -(X2, Y2), Lcolor
      Case 4 'Add a Step From First
        Line -Step(X2 - X1, Y2 - Y1), Lcolor, B
      Case 5 'Circle
        Pythagorean = (X2   X1) ^ 2 + (Y2 - Y1) ^ 2
        Hypotenuse = Sqr(Pythagorean)
        Radius = Hypotenuse
            Circle (X1, Y1), Radius, Lcolor
      Case 6 'Ellipse
        Radius = Sqr((X2 - X1) ^ 2 + (Y2 - Y1) ^ 2)
        For Aspect = 0 To 3 Step .1
        Circle (X1, Y1), Radius, Lcolor, , , Aspect
        Next Aspect
      Case 7 'Arc
        Radius = Sqr((X2 - X1) ^ 2 + (Y2 - Y1) ^ 2)
        RadianStart = 15 * (PI / 180)
        RadianEnd = 345 * .017453
        StartPt = RadianStart: EndPt = RadianEnd
        Circle (X1, Y1), Radius: DrawWidth = 4
        Circle (X1, Y1), Radius, Lcolor, StartPt, EndPt
        DrawWith = 1
      Case 8 'Pie
```

```
        Radius = Sqr((X2 - X1) ^ 2 + (Y2 - Y1) ^ 2)
        RadianStart = -15 * (PI / 180)
        RadianEnd = -345 * .017453
        StartPt = RadianStart: EndPt = RadianEnd
        Circle (X1, Y1), Radius: DrawWidth = 4
        Circle (X1, Y1), Radius, Lcolor, StartPt, EndPt
        DrawWith = 1
      Case 9 'Pie Slice
        Radius = Sqr((X2 - X1) ^ 2 + (Y2 - Y1) ^ 2)
        RadianStart = -345 * (PI / 180)
        RadianEnd = -15 * .017453
        StartPt = RadianStart: EndPt = RadianEnd
        Circle (X1, Y1), Radius: DrawWidth = 4
        Circle (X1, Y1), Radius, Lcolor, StartPt, EndPt
        DrawWith = 1
      End Select
DrawWidth = 4: PSet (X2, Y2): DrawWidth = 1
End If
  NextPoint = Not NextPoint
End Sub
```

X2=X:Y2=Y:这些值设定第二个屏幕坐标,该坐标代表直线的终点、方框的对角点或圆的半径点。

If NextPoint=True Then:仅当发生第二次鼠标键抬起事件时才触发下面的 Select/Case 语句。每当发生鼠标键抬起事件时,程序就用 Not 操作符(运算符)在 True 或 False 之间切换。

Select Case CurrentOption:CurrentOption 的值等于所选定的选项按钮的索引号。该值是在按钮 Option1 的单击(Click)事件中设置的,其设置方法是:CurrentOption=Index。

Case 0:选定了 Line(直线)选项按钮(索引号为 0)。

Line(X1,Y1)-(X2,Y2),Lcolor:从屏幕位置(X1,Y1)到屏幕位置(X2,Y2)画一条直线。**选项**:将线的颜色设成 Lcolor 的值。

Case 1:选定了 Box(方框)选项按钮(索引号为 1)。

Line(X1,Y1)-(X2,Y2),Lcolor,B:从屏幕位置 X1,Y1(方框的一个角)到屏幕位置 X2,Y2(对角)画一个方框。**选项**:用 Lcolor 的值设定方框的外框的颜色,用 B 指定方框形状。

Case 2:选定了 Box Filled(填充的方框)选项按钮(索引号为 2)。与上面的情况 Case 1 相同,只不过 Line 的选项改成了 BF,用于指定对方框进行填充。

Case 3:选定了 From Last X2,Y2(从上一点起)选项按钮(索引号为 3)。

Line-(X2,Y2),Lcolor:从上一点(Pset 语句设定)起画一条直线。

Case 4:选定了 Step 选项按钮(索引号为 4)。

Line-Step(X2-X1,Y2-Y1),Lcolor,B:另一种绘制方框形状的方法,该方法使用 Step 关键字。它加上相对于第一点(鼠标按下事件中 Pset 语句画出的点)的距离来绘制所需要的形状。

Case 5:选定了 Circle(圆)按钮(索引号为 5)。

Pythagorean：计算直角三角形两条较短的边。

Hypotenuse：计算上面平方和的平方根。

Radius：斜边就是圆的半径。

Circle(X1,Y1),Radius,Lcolor,,,Aspect：最后用 X1,Y1 画圆。

Case 6：选定了 Ellipse(椭圆)按钮(索引号为 6)。

Radius：这是前面的 Case 5 的简化计算方式。

For Aspect＝0 To 3 Step. 1：改变循环中每个圆的 Aspect(纵横比)值，每次递增 1。参见 Visual Basic 的参考指南中的 Circle Method(画圆方法)。

Case 7：选定了 Arc(圆弧)按钮(索引号为 7)。

RadianStart：**RadianEnd**：一个 360 度圆的弧度值。弧度数值从 3 点钟的位置算起，此处弧度值为 0。绘图过程中所用的弧度值在－6.2831853 到＋6.2831853 之间。

Case 8：选定了 Pie(圆饼图)按钮(索引号为 8)。

RadianStart, RadianEnd：为了用 Circle 语句画出一个饼图或扇形，在 Start(起始)和 End(终止)位置参数处都指定了负值。

Case 9：选定了 Pie Slice(圆饼扇形)按钮(索引号为 9)。

RadianStart, RadianEnd：为了用 Circle 语句绘制一个饼图或扇形，在 Start(起始)和 End(终止)位置参数处都指定了负值。

DrawWidth＝4:Pset(X1,Y1):DrawWidth＝1：首先将画线宽度设成 4 个象素宽，然后在鼠标按下的地方画一个点，最后将画线宽度重新设成默认值为 1 个象素宽。

NextPoint＝Not NextPoint：该语句将变量 NextPoint 的值从 True 设成 False，又从 False 设成 True，再从 True 设成 False，如此反复。这样就可以保证在发生第一次鼠标按下、抬起事件时设置 X1,Y1 的值，而在发生第二次鼠标按下、抬起事件时设置 X2,Y2 的值。

1.1.8　计算圆弧、饼图及扇形的弧度值

在绘制圆弧、饼图或扇形时，圆的起始和终止点都是以弧度度量的。一个弧度就是一个中心角，该角所截圆弧的长度等于该圆的半径。下面的表达式表示了圆与弧度的关系：

PI＝3.14159:**2＊PI**＝6.28318:**Circle**＝360 度

Radian(弧度)＝360/2＊PI:**Radian**(弧度)＝.017453

Degree(度)＝1/.017453:**Radians**(弧度)＝57.29577 度

Visual Basic 中的 Arcs(圆弧)和 Pies(饼图)＝2＊PI 或大约是 .017453＊360 度

在画圆弧、饼图或扇形时，0 弧度总在 3 点钟的位置。Circle 语句的 Start(起始)和 End(终止)值应该是介于－6.2831853 至＋6.2831853 之间的一个值。圆总是按顺时针方向绘制的，起始和终止值的正负会影响圆的显示方式(例如，圆、圆弧、饼图、扇形)。更详细的信息请参阅 Visual Basic 语言参考指南中的 Circle 语句。

1.1.9　Line 和 Circle 语句小结

Begin 程序给出了 Visual Basic 的 Line 和 Circle 语句的基本知识。每个图形的起始绘图位置都是在鼠标键按下时确定的，而实际图形的绘制则都是在鼠标键抬起时完成的。

Line 语句

Line 语句可分解为以下几部分：

1. 一条直线位于一个 X，Y 坐标网格内，该网格的默认原点 0，0 位于窗体或图形容器控制对象（例如图片框）的左上角。
2. 一条简单的直线可以通过画出该线的起点和终点坐标得到。起始点通常以 X1，Y1 表示，终止点通常以 X2，Y2 表示。起始点和终止点在网格上是按以下方式画出的：
   X1 是绘图点在 X 坐标轴上的位置（沿网格从左到右）。
   Y1 是绘图点在 Y 坐标轴上的位置（沿网格从上到下）。
   X2 是绘图点在网格中沿从左到右（水平）方向的点数。
   Y2 是绘图点在网格中沿从上到下（垂直）方向的点数。
   简单的公式是：Line(X1，Y1)－(X2，Y2)或（水平坐标，垂直坐标）－（水平坐标，垂直坐标）
3. Line 语句可以带若干选项。关键字 Step 可以用来画从 CurrentX，CurrentY 位置起始的直线。CurrentX 及 CurrentY 的默认位置是窗体或图形载体控制对象的左上角，即原点 0，0（水平坐标，垂直坐标）。
   执行了下列图形方法后，CurrentX 及 CurrentY 的设置就会改变：
   Circle：变为圆的圆心。
   Cls，EndDoc 及 NewPage：复位为 0，0。
   Line：变为直线的终点。
   Print：变为下一个打印位置。
   Pset：变为所画的点。
   简单的公式是：Line－Step(X2，Y2)或上一个 CurrentX，CurrentY（水平坐标，垂直坐标）。
   如果关键字 Step 被置于 X1，Y1 和 X2，Y2 之间，直线就从点(X1，Y1)开始，终止于 X2，Y2 处。简单表示如下：
   Line(X1，Y1)－Step(X2，Y2)
   或者
   (X1 水平坐标，Y1 垂直坐标)－（距 X1 的水平距离，距 Y1 的垂直距离）。
4. 直线也可以改为方框，方法是加上 B 或 BF 选项。只有在当前 FillStyle（填充式样）等于 1（默认值，该值使得方框的填充颜色是透明的）时，选项 B 才能使画出的方框只有外框，否则，画出的就是用纯颜色或图案来填充的方框。
   BF 选项使用窗体或图形控制对象的 ForeColor（前景颜色）填充方框。

Circle 语句

Circle 语句可分解为以下几部分：

1. 给出的圆位于一个(X，Y)坐标网格上，该网格的默认原点(0，0)位于窗体或图形控制对象（例如，图片框）的左上角。
2. 绘制一个简单的圆要确定圆心(X1，Y1)的位置。圆心坐标在网格上如下绘制：
   X1 是圆心在 X 轴上的坐标。

Y1 是圆心在 Y 轴上的坐标。

Circle 语句还必须包括一个半径值。半径就是从圆心(X1,Y1)到圆弧的距离(是圆直径的一半)。

一个简单的式子是:Circle(X1,Y1),50 或(圆心的水平坐标,圆心的垂直坐标),半径。

3. Circle 语句可以带若干选项。关键字 Step 可用来将一个圆置于 CurrentX,CurrentY 位置。CurrentX,CurrentY 默认位置是窗体或图形控制对象的左上角,亦即原点 0,0(水平坐标,垂直坐标)。

   每当执行下列图形方法时,CurrentX 和 CurrentY 的设置就会发生变化:

   Circle:变为圆对象的中心。

   Cls,EndDoc,NewPage:复位为坐标 0,0。

   Line:变为直线的终点(X2,Y2)。

   Print:变为下一个打印位置。

   Pset:变为所画的点。

   一个简单的式子是:Circle Step(X2,Y2)或上一个 CurrentX,CurrentY—(圆心的水平坐标,圆心的垂直坐标)。

   请注意,与 Line 语句不同,在画圆的关键字 Step 中没有使用连字符(-)。

5. Circle 语句的其他选项必须在半径后依次排列。如果你不使用某个选项,则必须在要插入选项之前的每个选项位置加入一个逗号(,)。

   Circle 语句中选项的次序是:

   Circle(X1,Y1),Color:RGB 值(例如,&H255)或 QBColor(0 到 15)圆的值。

   Circle(X1,Y1),,start,end:部分圆弧、饼图、扇形或椭圆形的弧度值。弧度值必须位于-6.2831853 与+6.2831853 之间。3 点钟的位置是 0 弧度。

   Circle(X1,Y1),,,,Aspect:圆的水平或垂直直径之间的比率。选用默认值为 1 时生成的是正圆。1 加上小数值(例如 1.01 或更大)生成一个较长的垂直椭圆,而小数值(例如 .01 或更大)生成一个较长的水平椭圆。

1.2 使用 Pset 语句

Pset 语句可用来将一个对象上的某个象素点设成指定的颜色。这里的对象指的是窗体、图片框或打印机。下面的 Pset 程序说明怎样通过数学算法生成直线和曲线。要想利用 Basic 语言直接画出直线或圆上的点,必须克服许多困难,正由于如此,Pset 程序绘制复杂对象的速度是相当慢的。(参见图 1.5)。

Pset 方法	**关键字**
Pset(X,Y)	Step:CurrentX 和 CurrentY
Pset(X,Y),QBcolor(4)	**选项**
Pset Step(X,Y),QBcolor(value)	Color:RGB 或 QBcolor

Pset 方法在 Visual Basic 程序中是非常有用的。除了在窗体上或图形控制对象上设置一

个彩色象素的这种常见用法外，它还可以用来替换 CurrentX、CurrentY 指针或用来生成自定义的点线或抖动的图案。本章中的 Pset 程序展示了在显示时钟面板上的数字时怎样对 Print 语句的结果进行偏移(清单 1.12)。Fill Pset 过程模拟在某个图形内部的颜色填充例程，而 Line Fill 则展示了这两个过程之间的速度差别。尽管 Pset 方法用在许多例程中，但它并没有得到任何赞赏，原因纯粹是因为速度问题。

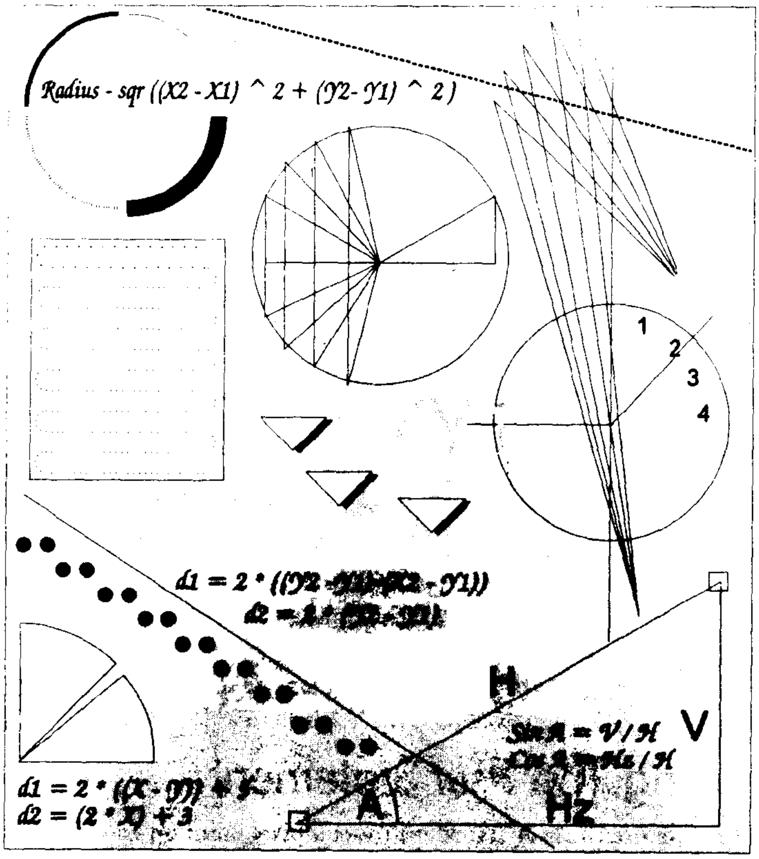


图 1.5　利用 Pset 方法绘制的多种图形

下面列出了 Pset 程序中所用的每一个命令按钮：

- Circle sin/cos：用 Sine(正弦)、Cosine(余弦)及 Pset 方法画一个由四个四分之一圆弧组成的圆。
- Circle Pset：用 Sine(正弦)、Cosine(余弦)及 Pset 方法按逆时针方向画一个圆。
- Circle Sqr：用 Sqr(平方根)函数和 Pset 方法画一个由四个四分之一圆弧组成的圆。
- Line Pset：用 Pset 方法画一条简单的水平线。
- Dash Pset：用 Pset 方法画一条水平虚线。
- Box Pset：用 Pset 方法画一个小方框形状。
- Grid Pset：用 Pset 方法画一个网格图案。
- Fill Pset：用 Pset 方法画一个填充图案。
- Line Pset：用 Line 方法画一个与前面的“Fill Pset”中相同的填充图案(比较执行速度)。

- Brensenham Line：用 Pset 方法在 45 度圆弧内画一组线。
- Brensenham 360：用 Pset 方法在 360 度圆弧内画一组线。
- Sine/Cosine：用 Sine 和 Cosine（余弦）函数画一根旋转的时钟指针，并用 Pset 方法确定钟面上数字的位置。
- Sine Cosine 2：用一个与上面 Sine/Cosine 中说明的程序类似的例程画若干饼图。

最后，任何时候单击鼠标右键都会画出一个移动的圆，该圆从随机放置的按钮控制对象弹回。Pset 程序的窗体及控制对象的属性在表 1.2 中列出。

表 1.2　Pset 窗体的属性

```
Object:Form
Object Name:Form1
Caption      = "Pset Drawing"       Height      = 6630
Left         = 225                  ScaleMode   = 3'Pixel
Top          = 1185                 Width       = 7455

Object:CommandButton
Object Name:Command1
Caption      = "SineCosine 2"       Height      = 510
Index        = 12                   Left        = 0
Top          = 5760                 Width       = 1770

Object:CommandButton
Object Name:Command1
Caption      = "Sine/Cosine"        Height      = 510
Index        = 11                   Left        = 0
Top          = 5280                 Width       = 1770

Object:Frame
Object Name:Frame1
Caption      = "Addition values"    Height      = 1575
Left         = 1860                 Top         = 0
Visible      =0'False               Width       = 1575

Object:OptionButton
Object Name:Option1
Caption      = "change X,Y"         Height      = 210
Index        = 3                    Left        = 60
Top          = 1200                 Width       = 1395

Object:OptionButton
Object Name:Option1
Caption      = "change Y"           Height      = 210
Index        = 2                    Left        = 60
Top          = 900                  Width       = 1215

Object:OptionButton
Object Name:Option1
Caption      = "change X"           Height      = 210
Index        = 1                    Left        = 60
Top          = 600                  Width       = 1215

Object:OptionButton
Object Name:Option1
Caption      = "circle"             Height      = 210
```

```
Index          = 0                   Left          = 60
Top            = 300                 Value         = -1 'Ture
Width          = 1215

Object:CommandButton
Object Name:Command1
Caption        = "Bresenham 360"     Height        = 510
Index          = 10                  Left          = 0
Top            = 4800                Width         = 1770

Object:CommandButton
Object Name:Command1
Caption        = "Bresenham Line"    Height        = 510
Index          = 9                   Left          = 0
Top            = 4320                Width         = 1770

Object:CommandButton
Object Name:Command1
Caption        = "Fill Line"         Height        = 510
Index          = 8                   Left          = 0
Top            = 3840                Width         = 1770

Object:CommandButton
Object Name:Command1
Caption        = "Fill Pset"         Height        = 510
Index          = 7                   Left          = 0
Top            = 3360                Width         = 1770

Object:CommandButton
Object Name:Command1
Caption        = "Circle Sqr"        Height        = 510
Index          = 2                   Left          = 0
Top            = 960                 Width         = 1770

Object:CommandButton
Object Name:Command1
Caption        = "Grid Pset"         Height        = 510
Index          = 6                   Left          = 0
Top            = 2880                Width         = 1770

Object:CommandButton
Object Name:Command1
Caption        = "Box Pset"          Height        = 510
Index          = 5                   Left          = 0
Top            = 2400                Width         = 1770

Object:CommandButton
Object Name:Command1
Caption        = "Dash Pset"         Height        = 510
Index          = 4                   Left          = 0
Top            = 1920                Width         = 1770

Object:CommandButton
Object Name:Command1
Caption        = "Line Pset"         Height        = 510
Index          = 3                   Left          = 0
Top            = 1455                Width         = 1770

Object:CommandButton
Object Name:Command1
```

Caption = "Circle Pset"　Height = 510
Index = 1　Left = 0
Top = 480　Width = 1770

Object:CommandButton
Object Name:Command1
Caption = "Circle sin/cos"　Height = 510
Index = 0　Left = 0
Top = 0　Width = 1770

图 1.6 给出的是 Pset 程序的主窗体，清单 1.9 列出了 Pset 使用的变量。

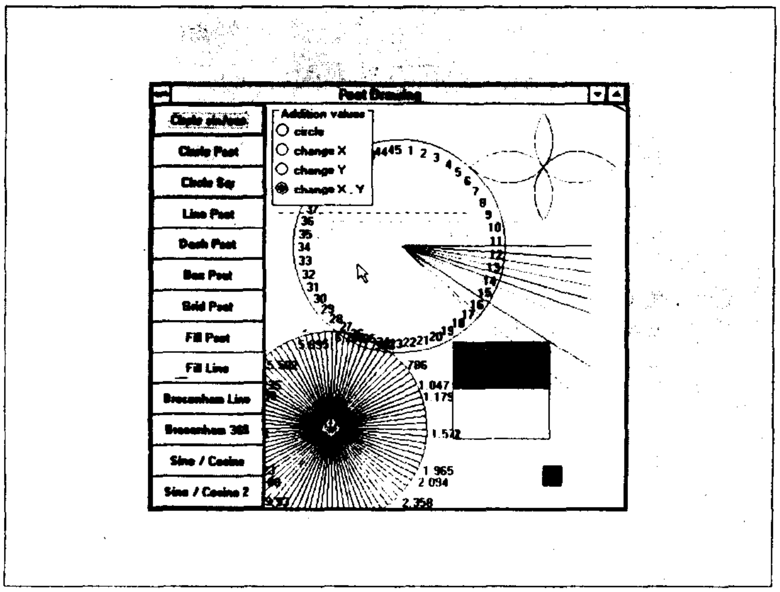

Pset Drawing
Circle sin/cos
Circle Pset
Circle Sqr
Line Pset
Dash Pset
Box Pset
Grid Pset
Fill Pset
Fill Line
Bresenham Line
Bresenham 360
Sine / Cosine
Sine / Cosine 2
Addition values
○ circle
○ change X
○ change Y
● change X , Y

图 1.6　Pset 程序的主窗体

清单 1.9　Pset 程序使用的变量

```
Declarations
Dim CurrentButton As Integer
Dim X1, Y1, X2, Y2 As Integer
Dim Yvalue,Xvalue As Integer
```

1.2.1　选择一个画图函数按钮

要想在窗体上绘制一个对象，请选定一个按钮，将鼠标在窗体上定位，然后单击。有关 Command1 Click 事件请参见清单 1.10。

清单 1.10　Command1_Click 事件

```
Sub Command1_Click(Index As Integer)
    Cls
    CurrentButton = Index
If Index<2 then Frame1.Visible=True Else Frame1.Visible=False
```

```
End Sub
```

Cls：擦除上一次显示在窗体上图象。

CurrentButton＝Index：存储赋给该按钮控制对象的索引号。该数值告知其他事件或过程选定的是哪个按钮。

If Index＜2 Then Frame1. Visible：当选定索引为 0 或 1 的按钮时，桌面上就会显示出一个带有若干绘图选项按钮的选择框控制对象。

1.2.2 在窗体上单击鼠标

Form_MouseDown 事件在清单 1.11 中给出。该事件执行与选定按钮相关联的绘图过程。

清单 1.11 Form_MouseDown 事件

```
Sub Form_MouseDown (Button%, Shift%, X As Single, Y As Single)
Tag = "Right Mouse"
If Button = 2 Then Exit Sub 'Bouncing Ball
If Command1(0).Left > 4 Then Exit Sub
Cls: ForeColor = QBColor(14 * Rnd)
Select Case CurrentButton
   Case 0
     Circle_sin_cos X, Y
   Case 1
     Circle_Pset X, Y
   Case 2
     Circle_Sqr X, Y
   Case 3
     Line_Pset X, Y
   Case 4
     Dash_Pset X, Y
   Case 5
     Box_Pset X, Y
   Case 6
     Grid_Pset X, Y
   Case 7
     Fill_Pset X, Y
   Case 8
     Fill_line X, Y
   Case 9
     X1 = X: Y1 = Y: X2 = X1 + 200: Y2 = Y1
     Brensenham X, Y
   Case 10
     X1 = X: Y1 = Y: X2 = X1 + 200: Y2 = Y1
     Brensenham_360 X, Y
   Case 11
     SineCosine X, Y
End Sub
```

Tag＝"Right Mouse"：如果单击了鼠标右键，所有 13 个按钮就会随机地放置在窗体上，

一个圆在窗体内与控制对象相碰撞。清单 1.24 中的 DO/LOOP 语句将根据 Tag="Right Mouse"标记将屏幕恢复成初始的形式。

Case 0 至 11:选定按钮的索引号与其中一个 Case 语句相匹配。12 个 Case 中的每一个都跳转至指定的过程,并将 X,Y 的当前值传递给相应的过程。

1.2.3　SineCosine 过程

清单 1.12 中给出的 SineCosine 过程是 Sine(sin)(正弦)和 Cosine(cos)(余弦)函数的一个很好的例子。改变 X,Y 的值或改变 A,B,C,D,E,F 或 G 变量的任一个数字值,就会得到一些异乎寻常的图案。SineCosine2 过程对前面提到的变量进行了设置。

SineCosine 过程说明了怎样用 Pset 语句对显示数字进行偏移。你还应当掌握使用 Sine(正弦)和 Cosine(余弦)处理角度的方法。当你读到第 6 章 Animator 程序时,就会更好地理解 SineCosine 和 SineCosine2 过程。第 6 章 Animator 程序广泛地应用了 Sin 和 Cos 语句。清单 1.12 给出了 SineCosine 过程。SineCosine 用正弦及余弦值画出一个带指针的表盘。

清单 1.12　SineCosine 事件

```
Sub SineCosine (X As Single, Y As Single)
Cls: DrawMode = 13
Circle (X, Y), 110, QBColor(4)
DrawMode = 6: Noon = 1
For R = 0 To 90 'to rotations
    A = Noon * .13819        'radians
    B = Sin(A)
    C = Cos(A)
    D = B * 100
    E = C * 100
    F - D + X
    G = (-1) * E + Y
    Line (X, Y)-(F, G)
      For delay = 1 To 8000: Next delay   'delay speed
    Line (X, Y)-(F, G) 'remark out to see divisions
    PSet (F - 10, G - 6): PSet (F - 10, G - 6)
      If Noon < 46 Then Print Noon
    Noon = Noon + 1
Next R
DrawMode = 13
End Sub
```

SineCosine 和 SineCosine2 采用了相同的算法。(SineCosine2 过程没有在本章中列出)。

For R=1 To 90:顺时针方向画的线数。

A=Noon * degrees:将 degrees(度数)换算成弧度数(除以 6.2831853)。每次循环 Noon(12 点钟的位置)加 1,这样就可以在表盘上沿顺时针方向得到相等的刻度。

B=Sin(A):根据 A 中存放的弧度值计算正弦值。

C=Cos(A):求出 A 的余弦值。

D=B * 100:本例中,100 代表表盘的水平宽度,改变该值将使表盘扩宽。

E=C * 100：本例中，100 代表表盘的垂直高度，改变该值将使表盘增高。

F=D+(X+0)：将 D 的值加上 X 坐标(表盘的中心)就可以得到表盘上表针的水平终点。将 0 改变为其他值就可以将表针沿水平方向拉长。

G=(−1) * E+(Y+0)：将 E 的值加上 Y 坐标(表盘的中心)就会得到表盘表针的垂直终点。将 0 改变为其他值就可以将表针沿垂直方向拉长。如果将 E 乘以(−1)，表针就沿顺时针方向旋转；如果将(−1)去掉，表针就从 6 点钟的位置开始，沿逆时针方向旋转。

Line(X,Y)−(F,G)：画表盘上的表针。

Noon=Noon+1：时间计数增 1。

Next R：重复上面的操作，计算表针下一次移动的结果。

计算 Sin(正弦)值

直角三角形的正弦是指角的对边与弦的比率。弦就是长的斜边，而对边通常就是直角三角形的垂直边。

假设角叫做 A，垂直边叫 Vert，斜边叫 Hyp，那么，A 的正弦值就是：

sin A=Vert/Hyp

或者说 sin A 等于三角形的垂直高度除以三角形的长斜边。

计算 Cos(余弦)值

余弦值是从直角三角形两条边的比率计算得来的。这个比率就是角的邻边长度除以斜边的长度。在大多数情况下，邻边指的是直角三角形的水平边。

如果角叫做 A，水平边叫做 Horiz，斜边叫做 Hyp，则 A 的余弦值就应该是：

Cos A=Horiz/Hyp

或者说 Cos A 等于三角形的水平长度除以三角形的长斜边。

1.2.4　Box　Pset 过程

清单 1.13 中给出的 Box　Pset 过程用一个嵌套的循环按象素点从左至右，再从右至左，同时垂直上升的顺序画出一个方框。这似乎并不十分新奇，因为 Line 语句可以更迅速地画出一个方框。不过在第 4 章中你将看到用这种方法可以在一个给定的宽度范围内查找到某种颜色。

清单 1.13　Box　Pset 过程

```
Sub Box_Pset(X As Single,Y As Single)
For R = -10 To 10 Step 1
    For RR = R -10 To 10 Step 1
        PSet (X + R, Y + RR)
    Next RR
Next R
End Sub
```

For R=−10 To 10 Step 1：这是嵌套循环的第一部分。该 For/Next 语句从−10 到 10

按步长 1 累加计数。内循环在每一列画出 20 行象素，而该语句画出 20 个象素列。

For RR=R-10 To 10 Step 1：这是嵌套循环的第二部分。该 For/Next 语句也是从-10 到 10 按步长 1 累加计数。该语句画出 20 行象素。

Pset(X+R,Y+RR)：根据下一次循环例程中指定的列及行位置画一个象素。

Next RR：内循环先结束。

Next R：外循环增 1，内循环重复执行。X 平移一列，Y 则确定下一列象素在屏幕上的位置。

1.2.5　Bresenham 方法

本节的某些例子中使用一些象素点来构成一个圆或一条斜线。当使用 Sin 或 Cos 来计算某个角度的斜度时，必须用小数进行处理。这些小数(通常称为浮点数)会显著地降低计算的速度。最好的方法是在任何可能的情况下都采用整数进行计算。

Bresenham 方法就是这样一种在屏幕上画象素点构成斜线的方法。大多数 Basic 中的 Line 和 Circle 语句都是采用从这种方法派生出来的方法计算这些形状的。你也许会发现，在图书馆里会找到大量的基于此方法的直线及曲线方程。这里给出的示例是最基本的，它只画出了从 0 度至-45 度的直线。清单 1.14 给出了 Bresenham 过程，它说明了怎样用整数公式来绘制一条斜线。

算法：

```
d1=2*((Y2-Y1)-(X2-X1))
d2=2*(Y2-Y1)
```

比较：

```
If D<0 Then
Pset 的 Y 值保持不变
If D>=0 then Pset 的 Y 值增 1
```

Bresenham 过程请见清单 1.14。

清单 1.14　Bresenham 过程

```
Sub Bresenham (X As Single, Y As Single)
    For N = 0 To 14
        D = 0: Y = Y1
        d1 = 2 * ((Y2 - Y1) - (X2 - X1))
        d2 = 2 * (Y2 - Y1)
            For R = X1 To X2
        If D < 0 Then D = D + d2 Else D = D + d1: Y = Y + 1
      PSet (R, Y), QBColor(N)
      Next R
    Y2 = Y2 + 12
    Next N
End Sub
```

清单 1.14 中的 Bresenham 公式只能计算 45 度直角空间内的直线，若要在整个 360 度

区域内画出斜线，只需将绘图空间分成 8 个象限，对四段中的每一段都重复该公式即可。

清单 1.15 中的 Bresenham_360 过程使用了与清单 1.14 中基本相同的 Bresenham 公式，只不过 X，Y 的值调换了一下。

清单 1.15 Bresenham_360 过程

```
Sub Bresenham_360 (X As Single, Y As Single)
For Pattern = 1 To 4 'the first four horizonal sections
  X1 = X: Y1 = Y: X2 = X + 200: Y2 = Y 'reset for each section
Select Case Pattern  'Choose statements could replace this Select Case
Case 1
   Angle = 1: Length = 200: S = 1
Case 2
   Angle = -1: Length = 200: S = 1
Case 3
   Angle = 1: Length = -200: S = -1
Case 4
   Angle = -1: Length = -200: S = -1
End Select

    For N = 0 To 14
      D = 0: YY = Y1
      d1 = 2 * ((Y2 - Y1) - (X2 - X1))
      d2 = 2 * (Y2 - Y1)
      For R = X To X + Length Step S
        If D < 0 Then D = D + d2 Else D = D + d1: YY = YY + Angle
        PSet (R, YY), QBColor(N)
      Next R
    Y2 = Y2 + 14
    Next N
Next Pattern
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'the four vertical sections, same code except Y values replace same X values
For Pattern = 5 To 8
    X1 = X: Y1 = Y: X2 = X + 200: Y2 = Y
Select Case Pattern
     Case 5
       Angle = 1: Length = 200: S = 1
     Case 6
       Angle = -1: Length = 200: S = 1
     Case 7
       Angle = 1: Length = -200: S = -1
     Case 8
       Angle = -1: Length = -200: S = -1
End Select

For N = 0 To 14
D = 0: YY = X1
d1 = 2 * ((Y2 - Y1) - (X2 - X1))
d2 = 2 * (Y2 - Y1)

      For R = Y To Y + Length Step S  'Y replaces X value
      If D < 0 Then D = D + d2 Else D = D + d1: YY = YY + Angle
      PSet (YY, R), QBColor(N)          'R and YY are reversed
```

```
        Next R
      Y2 = Y2 + 14
  Next N
  Next Pattern
  End Sub
```

For Pattern＝1 To 4：Bresenham 公式重复四次，生成 360 度范围内的直线。有关 Bresenham 公式的细节请参见清单 1.14 中给出的 Bresenham 过程。

1.2.6　Circle_Pset 过程

你可能想画一个圆，而又不用 Circle 语句。正如本程序前面所讲的那样，Pset 方法与 Basic 的 Circle 语句比较起来要慢一些。不过，画圆及转动直线的 Pset 过程可用来计算一个圆或直线角度的弧度(57.3)，度(360)，cos(余弦)，sin(正弦)，tan(正切)或 atn(反正切)。如果使用 Gage.VBX 及用前面提到的某个值来显示或设置 Gage.VBX 的指针，这将是很有用的。清单 1.16 中给出的 Circle_Pset 过程显示了一种设置圆上点的方法。

清单 1.16　Circle_Pset 过程

```
Sub Circle_Pset (X As Single, Y As Single)
PI = 3.141593
X1 = X: Y1 = Y
R = 75
For A = 0 To 2 * PI Step 0.9 / R
    Y = (R - Yvalue) * Sin(A)
    X = (R - Xvalue) * Cos(A)
    PSet (X1 + X, Y1 - Y)
        If Option1(0).Value = True Then Yvalue = 0: Xvalue = 0
        If Option1(1).Value = True Then Yvalue = Yvalue + .2
        If Option1(2).Value = True Then Xvalue = Xvalue + .6
        If Option1(3).Value = True Then Xvalue = Xvalue + .6:[IC:CCC]
    Yvalue = Yvalue + .3
Next A
End Sub
```

PI＝3.141593：PI 的值。

X1＝X:Y1＝Y：用来设置从圆心开始的偏移值。

R＝75：设定圆的半径。这与前一节中 Begin 程序用鼠标器设置该值是一样的。

For A＝0 To 2 * PI Step 0.9/R：A 代表正弦及余弦表中的一项，从 0(3 点钟所在的位置)到 360 度(2 * PI)就可以构成一个圆。关键字 step 使得每隔一定的角度设置一个点，改变该值可以改变点之间的间隔或者设置更多的点，而后者将会显著降低画圆的速度。

Y＝R * Sin(A):X＝R * Cos(A)：这两条语句计算相对于圆心 X1，Y1 的距离。

Yvalue 和 Xvalue 的值是通过按 Option 按钮来设定的，改变这两个值可以测试象素点的偏移位置。

Pset(X1＋X,Y1－Y)：现在，已经计算出了要设定的象素点相对于圆心的距离，可以将它们分别加上圆心的值 X1，Y1。

Next A：返回至开始处，重做一遍所有的操作。

1.2.7 Circle_sin_cos 过程

清单 1.17 中列出的 Circle_sin_cos 过程与前面的 Circle_Pset 过程是相同的，只不过将 PI 除以 2。这样就将圆分成 4 个相等的部分，你可以按自己的意愿画任意一段。在四个 Pset 代码行中的任意一行前面加上' 作为注释就可以只画出部分圆弧。

清单 1.17 Circle_sin_cos 过程，该过程分四段显示了一个圆

```
Sub Circle_sin_cos (X As Single, Y As Single)
PI = 3.141593:  X1 = X: Y1 = Y:  R = 75
  For A = 0 To PI / 2 Step .9 / R
    Y = (R - Yvalue) * Sin(A)
    X = (R - Xvalue) * Cos(A)
    PSet (X1 + X, Y1 - Y), QBColor(1)
    PSet (X1 - X, Y1 - Y), QBColor(2)
    PSet (X1 + X, Y1 + Y), QBColor(3)
    PSet (X1 - X, Y1 + Y), QBColor(4)
  Next A
End Sub
```

Y=R * Sin(A):X=R * Cos(A)：选定 Option1 按钮时，可以将 Yvalue 和 Xvalue 加上偏移值。改变这些值可以测试象素点的偏移位置。

1.2.8 Circle_Sqr 过程

用计算平方根的方法画一个圆时，并不能逐个象素地画到圆的直径的最左端和最右端。这是由于所用公式在 Y 值上的数学特性造成的。Y 值按一定的大小递减，该大小与 For/Next 语句中的重复次数有关。清单 1.18 中给出的 Circle_Sqr 过程用平方根公式画出了一个圆。

清单 1.18 Circle_Sqr 过程

```
Sub Circle_Sqr (X As Single, Y As Single)
XCenter = X: YCenter = Y
R = 75
  For XX = R To 0 Step -1
    Y = Sqr(R ^ 2 - XX ^ 2)
    PSet (XCenter + XX, YCenter - Y)
    PSet (XCenter + XX, YCenter + Y)
    PSet (XCenter - XX, YCenter + Y)
    PSet (XCenter - XX, YCenter - Y)
  Next
End Sub
```

Xcenter=X:Ycenter=Y：代表圆心，与 X1，Y1 相同。

R=75：设定圆的半径。你也许希望通过鼠标事件来设定该值。

For XX=R To 0 Step－1：XX 的值就是相对于圆心的偏移半径。本例中从 75 开始，一直减到 0。步长(step)值是－1，决定了倒计数顺序。

Y=Sqr(R＾2－XX＾2)：Y 的位置是变化的，每一个 Y 值确定了圆上的一个象素。我们采用直角三角形公式的最长边来获得变化的斜边终点，求出变化的直角三角形斜边的平方根后，我们并没有得到圆上每一点的 Y 坐标。因此利用该公式不能画出完整的圆。

Pset：以上的计算得出的圆弧是圆的四分之一。把 XX 和 Y 的值从负数改为正数，并从正数改为负数，就可以将此四分之一圆弧映射成一个整圆。

Next：返回到 For 语句，重复上述操作。

1.2.9　Dash_Pset 过程

Visual Basic 只提供了种类有限的点划线或虚线选项。改变下面过程中 Dash 和 Pixel 在 For/Next 循环中的长度，就可以得到多种不同式样线型。清单 1.19 用 Dash_Pset 过程说明了这一点。Dash_Pset 通过画象素来画虚线。

清单 1.19　Dash_Pset 过程

```
Sub Dash_Pset (X As Single, Y As Single)
X1 = X: Y1 = Y
PSet (X1 - 1, Y1)
  For DASH = 1 To 20
    For Pixel = 1 To 5
        PSet Step(1, 0)
      Next Pixel
      PSet Step(5, 0), BackColor
  Next DASH
End Sub
```

X1=X:Y1=Y：如果将 Pset 语句中的 X1，Y1 换成 X，Y，此句就不必使用了。

Pset(X1,Y1)：设定所要画线的起点。

For Dash=1 to 20：嵌套循环过程的第一层(或外层)循环。

For Pixel=1 to 5：嵌套循环的第二层(或内层)循环。

Pset Step(1,0)：在紧接上一次所画的点之后再画一个象素。

Next Pixel：重复 For DASH 循环，直到画出 5 个象素构成的一段短横线为止。

Pset Step(5,0),BackColor：在距上一次所画点之后 5 个象素远的地方设置下一个短横线的起点。

Next DASH：返回并重复上述所有操作。

1.2.10　Fill_Line 过程

Visual Basic 并未提供 DOS Basic 程序所具有的 Paint 语句。Paint 语句可像一把画刷一样填充屏幕的某一区域。Windows 应用程序接口(API)库中有许多填充简单矩形或复杂区域的方法。随着你继续阅读本书，你将会发现许多 API 画图函数的例子。同时，Fill_Pset 过程(清单 1.21)展示了一种相当慢的填充区域的方法，它与 DOS Basic 的 Paint 语句类似。为

了演示的目的，Fill_Line 过程中用了一条简单的 Line 语句循环来重复 Fill_Pset 过程中同样的填充效果。清单 1.20 中首先列出了 Fill_Line 过程。

清单 1.20　Fill_Line 过程，它在速度上优于 Fill_Pset 过程

```
Sub Fill_line (X As Single, Y As Single)
Line (X, Y)-(X + 100, Y + 100), QBColor(4), B
  For RR = 1 To 50 Step 2
    Line (X + 1, Y + RR)-(X + 100, Y + RR)
  Next RR
End Sub
```

Line(X,Y)-(X+100,Y+100),QBColor(4),B：画一个红色的矩形框。

For RR=1 To 50 Step 2：For/Next 语句按步长 2 从 1 计数到 50，也就是每隔两个象素单位设置一个 Y 坐标。

Line(X+1,Y+RR)-(X+100,Y+RR)：画出红色矩形内的每一条线。如果将 For/Next 语句中的 Step 2 关键字去掉，方框的上半部分就会填充成实心的。

1.2.11　Fill_Pset 过程

清单 1.21 中的 Fill_Pset 过程采用了一种通过画单个象素的方法来画直线。象素线从左向右延伸并在绘制过程中不断向前查看邻近象素的颜色。如果在路径上的下一个绘图点邻近象素的颜色与方框外框的颜色相同，则进行以下操作：

1. 象素线向上移动一定的距离，该距离由 Spacing 变量给定。
2. 象素线现在向相反方向移动(从右到左)。
3. 该处理过程不断重复，直至边界区域被填充完为止。

当象素线要越过窗体的边界时，必须终止该程序。否则象素会越出屏幕，直至达到整数最大值。在 Fill_Pset 过程(如清单 1.21 所示)中，为防止象素越过屏幕边界，循环中对 NextX 和 CurrentX 的值进行了检测。

清单 1.21　Fill_Pset 过程通过设置象素的方法画出了一个填充图案

```
Sub Fill_Pset (X As Single, Y As Single)

Line (X - 50, Y - 50)-(X + 50, Y + 50), QBColor(4), B

CurrentX = X: NextX = X + 49
CurrentY = Y: NextY = Y
N = -1: Spacing = 2

Do
  If Point(CurrentX + N, CurrentY) = QBColor(4) Then
    NextX = NextX - N: NextY = NextY - Spacing
    Direction = Not Direction
    If Direction = 0 Then N = -1 Else N = 1
  End If

  If NextY <= (Y - 50) Or NextY >= (Y + 50) Then Beep: Exit Do
```

```
    If CurrentX < 0 Or CurrentX > ScaleWidth Then Exit Do
    PSet (NextX, NextY), QBColor(1): NextX = CurrentX - N
  Loop
  End Sub
```

Line(X－50,Y－50)－(X＋50,Y＋50),QBColor(4),B:画一个红色的矩形框。

N＝－1:N 的值代表了计数方向。－1 按降序计数,而＋1 按升序计数。该值从正转换成负(或反过来)以控制直线延伸的方向。

Spacing:改变该值以控制象素线的间隔。正值使得象素向上移动,而负值使得象素向下移动。

DO:循环执行以下的语句。

If Point(CurrentX＋N,CurrentY)＝QBColor(4) Then:如果位于当前所设置象素正前方的象素是红色的,那么就必须改变方向,因为已达到了红色方框的边界。

NextX＝NextX－N:NextY＝NextY－Spacing:NextX 和 NextY 代表下一个要设置的象素点位置。

Direction＝Not Direction:每当象素线达到方框的边缘时,Direction 变量的值就会切换成 True 或 False。

If Direction＝0 Then N＝－1 Else N＝1:如果 Direction 变量变为 False(0),则象素从右向左延伸;如果 Direction 变量变为 True(－1),则象素从左向右延伸。

If NextY:If CurrentX:如果象素移动距离超出了指定的范围,循环就将终止,过程随之结束。

Pset(NextX,NextY):根据 NextX,NextY 的值设置每一个象素点。可试着将 QBColor(1)改为 QBColor(Rnd * 15),这时方框内的各点将设置为不同的颜色。

NextX＝CurrentX＋N:加上或减去 N 以算出下一个象素的位置。

Loop:继续上述循环,直至上述条件为真时终止。

1.2.12　Grid_Pset 过程

清单 1.22 中给出的 Grid_Pset 过程与 Box_Pset 过程基本相同,所不同的是象素点之间的距离和循环所用的步长(step)稍大一些,因此 Pset 点也就隔得更远一些。这是一个给程序加上网格点的很好的小过程。

清单 1.22　Grid_Pset 过程

```
Sub Grid_Pset (X As Single, Y As Single)
  For R = -90 To 90 Step 15
    For RR = -90 To 90 Step 15
        PSet (X - R, Y - RR)
    Next RR
  Next R
End Sub
```

1.2.13　Line_Pset 过程

清单 1.23 中列出的 Line_Pset 过程就像它本身一样简单。当用 Pset 语句画线时，你会发现与 Line 语句比较起来是多么缓慢。不过有时你可能想要显示一条缓慢延伸的线。

清单 1.23　Line_Pset 过程

```
Sub Line_Pset (X As Single, Y As Single)
Y1 = Y
  For R = X To X + 200
    PSet (R, Y1)
    Y1 = Y1 + 0
  Next R
End Sub
```

For R＝X To X＋200：R 的值从当前鼠标键按下的位置 X 开始，一直增加 200 个象素单位。

Pset(R，Y1)：沿着 X 轴(从左至右)设定每个象素点。

Y1＝Y1+0：改变 Y1 的值，就可以得到一条斜线，但是如果它与水平或垂直方向的夹角不是 45 度，则象素点看上去是扭曲的。生成斜线的简单方法是不断地将起始点和终止点除以 2，由于这要涉及浮点数运算，因此要比你现在所看到的慢得多。

1.2.14　Visual Basic 的 Point 语句

Point 语句和 Pset 语句相似，只不过 Point 语句不是画象素而是返回给定的 X 和 Y 坐标处象素点的 RGB 颜色。如果给定的 X，Y 坐标在窗体之外或某个控制对象位于 X，Y 坐标处，则 Point 语句返回－1。清单 1.24 中的 Form_MouseUp 事件演示了使用 Point 语句的方法。当单击鼠标右键后，Point 语句就不停地检查位于移动圆正前方的象素点，看它是否等于－1。如果象素值等于－1，那么说明移动的圆碰到了某个按钮控制对象，或者达到了窗体的边界。该圆就向相反的方向移动，并在遇到其他对象时再次弹开。

清单 1.24　查找一个象素的颜色值

```
Sub Form_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Yvalue = 0; Xvalue = 0
'''''''''''''''''''''

If Button = 2 Then 'Bouncing Ball
    color = -1: r = 10     'radius
    X1 = 1: Y1 = 1: Cls    'distance/angle
    X = ScaleWidth - 20    'starting point
    Tag = ""               'escape from loop flag
For r = 0 To 12
  Command1(r).Left = (Rnd * ScaleWidth \ 2)
Next r
  Do
    If Point(X + r, Y) = color Or Point(X - r, Y) = color Then
        X1 = -X1
    End If
    If Point(X, Y + r) = color Or Point(X, Y - r) = color Then
        Y1 = -Y1
    End If
    X = X + X1
    Y = Y + Y1
    Circle (X, Y), r, QBColor(4)   'red
        For rr = 1 To 1000: Next rr 'speed
    Circle (X, Y), r, QBColor(15) 'white
DoEvents 'check for button click to exit loop
    If Tag = "Right Mouse" = True Then
        For r = 0 To 12
            Command1(r).Left = 4
        Next r
      Exit Do
    End If
  Loop
End If
End Sub
```

Yvalue:Xvalue：设置起始坐标的默认值。

If Button=2 Then：如果单击了鼠标右键，则执行下列语句。

Color=-1:r=10：用于检测 Point 语句的返回值是否为-1，圆半径为 10 个象素点。

X1=1:Y1=1:cls：增加或减少 X1，Y1 的值就会改变圆的距离和角度。

X=ScaleWidth-20：圆的起始坐标。

Tag=""：该标志用来控制从循环中跳出。

Command1(r).Left：将 13 个按钮随机地放置在窗体上。

DO：执行以下的代码段，直至按下了一个按钮为止。

If Point(X+r,Y)=Color：检查移动中的圆正前方象素值，如果返回值是-1，则改变圆的移动方向。

Circle(X,Y),r,QBColor(4)：画一个红色的圆。

For rr=1 To 1000：控制圆的移动速度。

Circle(X,Y),r,QBColor(15)：擦掉红色的圆。

DoEvents：检查是否单击了 13 个按钮中的某一个。

If Tag="Right Mouse"：单击 13 个按钮中的任一个都会将 Tag 的值改为“Right Mouse”，循环过程将因此而停止绘制圆形操作并退出鼠标键抬起事件过程。

Loop：继续 Do/Loop 循环，直至单击了某个按钮为止。

1.2.15 Pset 语句小结

Pset 语句不只是可以用来将某个对象上的点设成指定的颜色，还可以用来设置窗体中的象素点。Pset 也可代替 CurrentX 和 CurrentY 来调整文字输出的位置，但是，如果未将绘图式样属性改为不可见或未将象素颜色设成背景颜色，屏幕或页面上就会只显示出一个点。

本书中有几个程序就是用 Pset 方法来设置可调整的标尺数字的。还应该提醒一句，通过设置窗体或图形控制对象的 DrawWidth（绘图宽度）属性可以增大所显示点的大小。

与 Line 和 Circle 语句类似，Pset 方法也可以使用 Step 关键字。其原理与本章“Line 和 Circle 语句小结”一节中介绍的一样，读者可参阅。简单的形式是：Pset Step(X,Y)或上一个 CurrentX,CurrentY-(水平坐标，垂直坐标)。请注意，与 Line 语句不同的是，Pset 的 Step 关键字中没有使用连字符(-)。

1.3 移动图形对象

本节介绍几种用拖放方式将绘制的图形从屏幕的一部分移到屏幕另一部分的方法。在 Visual Basic 参考指南的“对鼠标器事件作出响应”中给出了一个用鼠标设定一个控制对象的位置的例子。不过该例子在移动过程中只是将控制对象的中心定位于鼠标的当前 X,Y 位置。大多数演示程序都可以从鼠标按下的点开始移动对象，给人一种印象是在平滑地移动的错觉。图 1.7 演示了怎样使一个对象看上去像是在屏幕上移动。

本节给出了几种获得这种平滑移动效果的方法，其中包括一次移动几个对象并使这些对象在移动的过程中一直显示在屏幕上。由于当一个对象正在移动时重画该对象要做大量的工作，因而我们在此只移动对象的外框。如果要移动一个全彩色图形而且要使该图形在移动中仍然可见，就必须求助于 Windows 的 API 屏幕捕捉函数。如果使用的是 Visual Basic 4.0，则可用 PaintPicture 方法来传输屏幕和图片图象。

PaintPicture 类似于 Windows BitBlt 或 StretchBlt API 函数。有关 API 函数 BitBlt 的例子可以在附带软盘 16MISC 目录下的 Invert,C-Clip 和 VB-Clip 程序中找到。

表 1.3 列出了 Movetest 程序中窗体和控制对象的属性。

表 1.3 Movetest 窗体的属性

Object:Form
Object Name:Movetest
Caption = "Moving objects"   DrawMode = 6 'Invert
Height =4950   Left = 720
ScaleMode =3' Pixel   Top = 1845
Width = 6975

Object:CommandButton
Object Name:Command1
Caption = "Exit"   Height = 435
Index = 12   Left = 4140
Top = 4140   Width = 1395
Object:CommandButton
Object Name:Command1
Caption = "T/B Stretch"   Height = 435
Index = 11   Left = 2760
Top = 4140   Width = 1380

Object:CommandButton
Object Name:Command1
Caption = "Slide Stretch2"   Height = 435
Index = 10   Left = 1380
Top = 4140   Width = 1395

Object:CommandButton

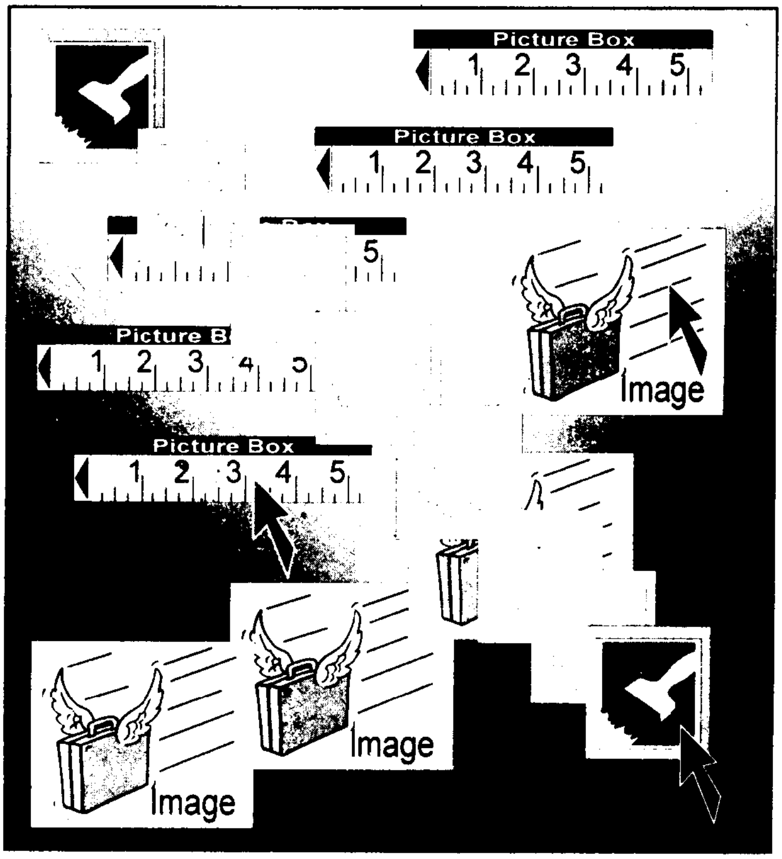

Picture Box
1 2 3 4 5
Picture Box
1 2 3 4 5
5
Picture B
1 2 3 4 5
Picture Box
1 2 3 4 5
Image
Image
Image

图 1.7　对象看上去像是在屏幕上移动

Object Name:Command1
Caption = "Slide Stretch1"　Height = 435
Index = 9　Left = 0
Top = 4140　Width = 1380

Object:CommandButton
Object Name:Command1
Caption = "Blank"　Height = 435
Index = 8　Left = 4140
Top = 3720　Width = 1380

Object:CommandButton
Object Name:Command1
Caption = "L/R Stretch"　Height = 435
Index = 7　Left = 2760
Top = 3720　Width = 1395

Object:CommandButton
Object Name:Command1
Caption = "Slide T/B"　Height = 435
Index = 6　Left = 1380
Top = 3720　Width = 1380

Object:CommandButton
Object Name:Command1
Caption = "Slide L/R"　Height = 435

```
Index          = 5              Left           = 0
Top            = 3720           Width          = 1395

Object:CommandButton
Object Name:Command1
Caption        = "Move Objects"  Height         = 435
Index          = 4              Left           = 5520
Top            = 0              Width          = 1395

Object:CommandButton
Object Name:Command1
Caption        = "XY Move"      Height         = 435
Index          = 3              Left           = 4140
Top            = 0              Width          = 1380

Object:CommandButton
Object Name:Command1
Caption        = "Cls Screen"   Height         = 855
Index          = 5520           Left           = 3720
Width          = 1395

Object:CommandButton
Object Name:Command1
Caption        = "Offset shift" Height         = 435
Index          = 2              Left           = 2760
Top            = 0              Width          = 395

Object:CommandButton
Object Name:Command1
Caption        = "Center shift" Height         = 435
Index          = 1              Left           = 1380
Top            = 0              Width          = 1380

Object:CommandButton
Object Name:Command1
Caption        = "Draw Box"     Height         = 435
Index          = 0              Left           = 0
Top            = 0              Width          = 1395
```

图 1.8 给出了 MoveTest 程序的一个屏幕画面，清单 1.25 则列出了 MoveTest 使用的变量。

清单 1.25 MoveTest 程序中使用的变量

```
Declarations
Dim X1, Y1, X2, Y2 As Integer
Dim XX1, YY1, XX2, YY2 As Integer
Dim ShiftX, ShiftY As Integer
Dim WB, NB, EB, SB As Integer
Dim N As Integer
Dim CurrentButton As Integer
```

X1,Y1,X2,Y2;XX1,YY1,XX2,YY2:临时边界框位置值。

ShiftX,ShiftY:存储对象移动的位移量。

WB,NB,EB,SB:角边框位置(西,北,东,南)。

N:存储偏移倾斜值。

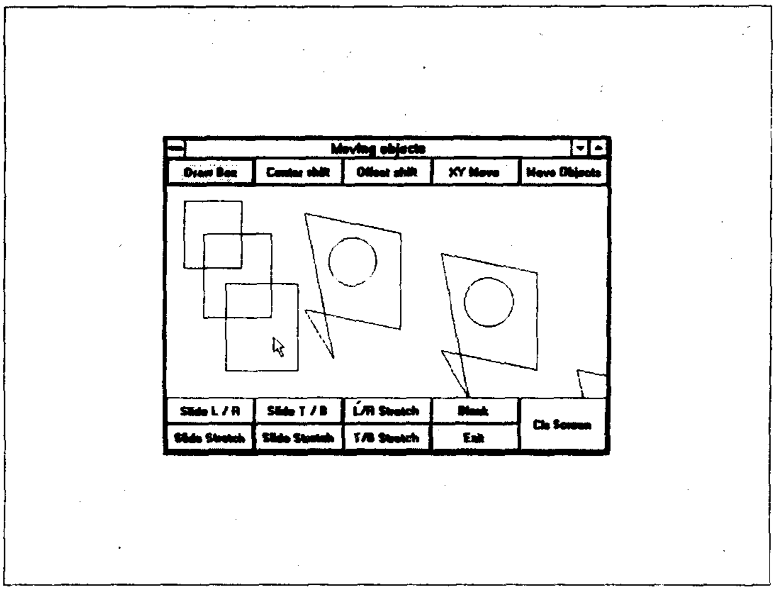

Moving objects
Draw Box
Center shift
Offset shift
XY Move
Move Objects
Slide L / R
Slide T / B
L/R Stretch
Mask
Cls Screen
Slide Stretch
Slide Stretch
T/B Stretch
Exit

图 1.8　MoveTest 程序的一个屏幕画面

CurrentButton：存放当前选定按钮的索引号。

1.3.1　单击 Cls 按钮

ClsBtn Click 事件重新设置本程序中变量的默认值。该例程在清单 1.26 中给出。

清单 1.26　ClsBtn_Click 事件

```
Sub ClsBtn_Click ()
    Cls
    X1 = 0: Y1 = 0: X2 = 0: Y2 = 0
    XX1 = 0: YY1 = 0: XX2 = 0: YY2 = 0
End Sub
```

Cls：清除屏幕，擦去所有的图象。

X1、Y1、X2、Y2：重新设置下次移动的值。

1.3.2　选定一个移动按钮

单击一个按钮就会存储当前选定的移动选项，该功能是通过清单 1.27 中列出的 Command1 Click 事件来完成的。图 1.9 展示了主要按钮的功能。

清单 1.27　Command1_Click 事件

```
Sub Command1_Click (Index As Integer)
    CurrentButton = Index
    If Index = 12 Then End
End Sub
```

CurrentButton：等于当前选定的按钮。

DrawMode 属性设置

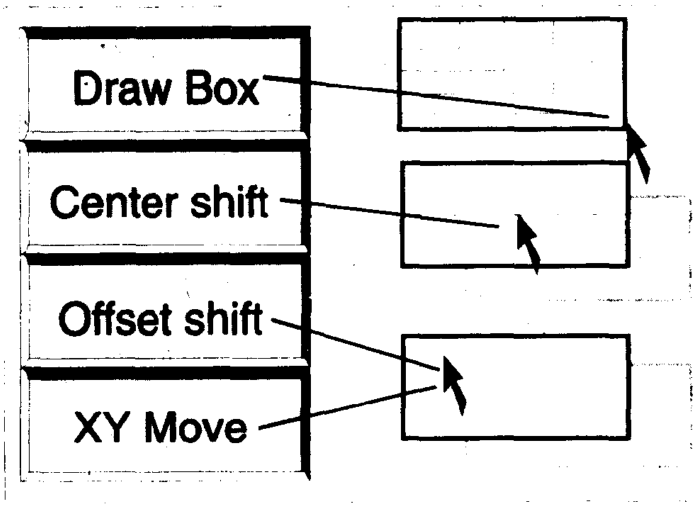


图 1.9 主要按钮的功能

1. Black Pen
2. notMergePen
3. Mask Not Pen
4. Not Copy Pen
5. Mask Pen Not
6. Inverse
7. Xor Pen
8. Not Mask Pen
9. Mask Pen
10. Not Xor Pen
11. Nop
12. Merge Not Pen
13. Copy Pen
14. Merge Pen Not
15. Merge Pen
16. White Pen

1.3.3 Form_MouseDown 事件

在移动某个对象时，应该改变绘图模式(Draw Mode)，这样所画的第一个对象就会被下一次鼠标移动调用擦掉。如果在鼠标移动(Mouse Move)事件中放置了一条绘图语句，那么鼠标每移动一次就会将该对象画一次。在正常模式(13)下，如果将鼠标从窗体的一边移到另一边，就会画出成千上万个这种对象。在一个对象上面再画一个完全相同的对象，就会给人一种动画的感觉。达到这种效果的方法之一就是将绘图模式(Draw Mode)设成 Inverse(6)(反转)，该模式反相绘制象素。例如，如果对象是一个黑色的外框方框，则在它上面再画一个同样的方框就会将该方框擦掉，这是因为将黑色象素反转就是白色象素，而白色象素也就是背景色。注意：DrawMode(7)X_or 是用来擦除具有多种背景色的对象的。清单 1.28 给出了 Form_MouseDown 例程，该例程对边界方框及鼠标的位置进行了初始化设置。

清单 1.28 Form_MouseDown 事件

```
Sub Form_MouseDown (Button%, Shift%, X As Single, Y As Single)
Drawmode = 6
Select Case CurrentButton
  Case 0 'Draw Box
    Cls
    X1 = X: Y1 = Y: X2 = X1: Y2 = Y1
  Case 1 'Center Box on mouse
    XX1 = 0: YY1 = 0: XX2 = 0: YY2 = 0 'reset
  Case 2 'Shift box position with mouse
    ShiftX = X - ((X1 \ 2) + X2 \ 2)
    ShiftY = Y - ((Y1 \ 2) + Y2 \ 2)
    XX1 = ShiftX: YY1 = ShiftY
  Case 3 'Move box from X and Y offset
    XX1 = X: YY1 = Y
    WB = X1: NB = Y1: EB = X2: SB = Y2
  Case 4 'Move multi objects
   Cls
   N = 20 'Skew
   X1 = X: Y1 = Y: X2 = X1: Y2 = Y1
    Line (X1, Y1)-Step(100, 0 + N)
    Line -Step(0, 100)
    Line -Step(-100, 0 - N)
    Line -Step(30, 50)
    Line -Step(-30, -150)
    Circle (X1 + 50, Y1 + 50), 25
  Case 5 To 12  'other effects
    XX1 = X: YY1 = Y
    WB = X1: NB = Y1: EB = X2: SB = Y2
End Select
End Sub
```

DrawMode=6：将绘图模式改为 Inverse(反转)。

Select Case CurrentButton：根据所选定的按钮执行下面相应的操作。

Case 0：将方框的临时角点值设成都等于鼠标当前 X，Y 位置。

Case 1：将方框位置、鼠标位置偏移值都重新设回默认值 0。

Case 2：变量 ShiftX 是求出方框的水平中心点，而变量 ShiftY 则求出方框的垂直中心点。当要在一个窗体上显示一个方框时，该公式用来算出此方框准确的中心点位置。这个公式也可以写成 ShiftX=X-(X1+X2)/2。但是如果使用 Twip ScaleMode 模式，在将 X1 和 X2 相加时可能会造成 ShiftX 整数溢出。这样就应该将 ShiftX 和 ShiftY 整数声明为双精度(Double)数据类型，这将占用更多的内存。

XX1，YY1：存储 ShiftX 和 ShiftY 的值，在鼠标移动事件中用作偏移值。

The XX1 And YY1 Set：当前 X，Y 位置的临时存储单元。

WB=X1：NB=Y1：EB=X2：SB=Y2：本书中大部分程序都用 W，N，E，S 变量名来存储边界方框的角点位置，而 X1，Y1，X2，Y2 几乎总是用来临时存放 X，Y 的值。

Case 4：该代码段在一个由多条线组成的对象内画出一个圆。首先在鼠标按下事件中画出图象以便擦去或设定新的图象。由于使用反转 Draw Mode 绘图，每移动一次鼠标，图象就会被设成黑色，然后再设成白色。如此连续不断地进行下去，就会给人一种对象在移动的感觉。

Case 5 to 12：这些值与 case 3 中设置的值相同。

附注：　*在屏幕上用鼠标抬起、移动和按下事件来移动对象的原理将在第 2 章“Artisan 程序”清单 2.7 中详细讨论。*

如果采用上述的某些公式来移动 Visual Basic 的控制对象，就不必加上 X2，Y2 的值。因

为 Visual Basic 在绘制这些控制对象的过程中，已经考虑了 X2，Y2 的值。

1.3.4 Form_MouseMove 事件

当用鼠标在窗体上拖动对象时，鼠标移动事件就可用来控制该对象的移动位置。移动对象的关键是在对象的当前坐标上加上一个偏移值。对于 XYmove 例程来说，这个偏移值是 Xcenter 变量；而如果使用 Center Shift 或 Offset Center 例程，则该偏移值就是 ShiftX，ShiftY。偏移值表示的是移动前的边界方框 X1，Y1－X2，Y2 初始中心坐标减去鼠标指针 X，Y 位置的距离。“边界方框”是用来描述完全包住所移动对象区域的术语。边界方框形状是矩形的，度量单位与 Line 语句方框相同。

附注： 一个边界方框是（X1，Y1）－（X2，Y2），（左，上）－（右，下）或（西，北）－（东，南）。

XYmove 例程中使用了一种更简单的计算偏移值的方法。在发生鼠标键按下事件期间，XYmove 存储鼠标键按下时的坐标及边界方框的角点位置，当发生鼠标移动事件时，XYmove 就减去从初始值移动的距离。在清单 1.29 给出的 Form_MouseMove 过程中可以看到这种方法。Form_MouseMove 过程根据鼠标的移动画出边界方框。

清单 1.29 Form_MouseMove 事件

```
Sub Form_MouseMove (Button%, Shift%, X As Single, Y As Single)
If Button Then
    XCenter = XX1 - X: YCenter = YY1 - Y
    ShiftX = X - ((X1 \ 2) + X2 \ 2)
    ShiftY = Y - ((Y1 \ 2) + Y2 \ 2)

Select Case CurrentButton
  Case 0        'Draw box
    Line (X1, Y1)-(X2, Y2), , B
        X2 = X
        Y2 = Y
    Line (X1, Y1)-(X2, Y2), , B
  Case 1 To 2 'Center shift & Offset shift -Mouse
    Line (X1, Y1)-(X2, Y2), , B
    X1 = (X1 - XX1) + ShiftX
    Y1 = (Y1 - YY1) + ShiftY
    X2 = (X2 - XX1) - ShiftX
    Y2 = (Y2 - YY1) + ShiftY
    Line (X1, Y1)-(X2, Y2), , B
  Case 3        'XY Move -Mouse
    Line (WB, NB)-(EB, SB), , B
    WB = X1 - XCenter
    NB = Y1 - YCenter
    EB = X2 - XCenter
    SB = Y2 - YCenter
    Line (WB, NB)-(EB, SB), , B
  Case 4        'Move Object - Mouse
    Line (X1, Y1)-Step(100, 0 + N)
    Line -Step(0, 100)
    Line -Step(-100, 0 - N)
```

```
      Line -Step(30, 50)
      Line -Step(-30, -150)
      Circle (X1 + 50, Y1 + 50), 25
      X1 = X:Y1 = Y
    Line (X1, Y1)-Step(100, 0 + N)
    Line -Step(0, 100)
    Line -Step(-100, 0 - N)
    Line -Step(30, 50)
    Line -Step(-30, -150)
    Circle (X1 + 50, Y1 + 50), 25
  Case 5         'Slide Left/Right
      Line (WB, NB)-(EB, SB), , B
      WB = X1 - XCenter
      EB = X2 - XCenter
      Line (WB, NB)-(EB, SB), , B
  Case 6         'Slide Top/Bottom
      Line (WB, NB)-(EB, SB), , B
      NB = Y1 - YCenter
      SB = Y2 - YCenter
      Line (WB, NB)-(EB, SB), , B
  Case 7         'Left/Right Stretch
      Line (WB, NB)-(EB, SB), , B
      NB = (Y1 - Y) - (YCenter - YY1)
      WB = X
      Line (WB, NB)-(EB, SB), , B
  Case 8
      'Test your own here
  Case 9         'Slide Stretch (L/R)
      Line (WB, NB)-(EB, SB), , B
      WB = X
      NB = Y
      EB = X2 - XCenter
      Line (WB, NB)-(EB, SB), , B
  Case 10        'Slide Strech (T/B)
      Line (WB, NB)-(EB, SB), , B
      WB = X
      NB = Y
      SB = Y2 - YCenter
      Line (WB, NB)-(EB, SB), , B
  Case 11        'Top / Bottom Stretch
      Line (WB, NB)-(EB, SB), , B
      WB = (X1 - X) - (XCenter - XX1)
      SB = Y
      Line (WB, NB)-(EB, SB), , B
  Case 12
      'Test your own here
End Select
End If
End Sub
```

If Button Then：仅在拖动过程中鼠标键被按下时才继续执行以下的代码。

XCenter＝XX1－X：YCenter＝YY1－Y：如果选定了 Center Shift 按钮(索引号为 1)，方框中心就被设置于鼠标指针所在位置，因此，如果在鼠标按下事件发生时，XX1，YY1

偏移值等于 0，该方框中心就被设置于鼠标指针所在位置。

如果选定了 Offset Shift 按钮（索引号为 2），方框中心也被置于鼠标指针处，但是接着又被加上 XX1，YY1 偏移值移回原来的位置。

ShiftX，ShiftY：该公式与鼠标键按下事件中计算对象边界方框中心位置的公式相同。

Select Case CurrentButton：根据选定的按钮，执行相应的代码。

Case 0：画出方框。有关画一个可移动方框图象的细节请参见第 2 章中的 ArtWork_MouseMove 事件。

Case 1 To 2：如果选定了 Center Shift 按钮（索引号为 1），方框中心就被设置于鼠标指针的当前位置，拖动鼠标时，方框中心总是位于指针所在位置。如果选定了 Offset Shift 按钮（索引号为 2），就从方框的当前位置开始移动，而不用将方框中心置于鼠标指针处，方框还将随着鼠标指针一起移动。要达到这种效果，可以计算出从当前方框的中心到鼠标指针位置的偏移值，这些偏移值就是鼠标按下事件中的 XX1 和 YY1 的值。Artisan 程序使用的就是这种方法。

Case 3：XYMove 按钮（索引号为 3）执行与 Offset Shift 按钮相同的操作。不过，计算公式有所不同，它不需在鼠标按下事件中计算任何偏移值，如若要存储新位置的值，就必须在 Mouse Up（鼠标抬起）事件中来完成。这样使得代码较短。第 15 章中的 ArtAPI 程序使用了这种方法。

Case 4：简单地将对象画两次，第二幅图象擦掉第一幅，如此往复。必须首先在鼠标按下事件或某个与鼠标按下调用相连系的过程中画出该对象。第 15 章中的 ArtAPI 程序采用了此方法绘制多种形状的对象。

Case 5 To 12：这些代码段展示了多种移动对象的方法。多数的位置值对执行相同绘图函数是可互换的。

1.3.5 Form_MouseUp 事件

对象放下后，必须复位所有的边界方框位置，并对各个变量进行总清除。清单 1.30 中给出的 Form_MouseUP 例程将所有的边界方框位置值进行了复位。

清单 1.30 Form_MouseUp 事件

```
Sub Form_MouseUp (Button%, Shift%, X As Single, Y As Single)
Select Case CurrentButton
   Case 3, 5 To 12
     X1 = WB: Y1 = NB: X2 = EB: Y2 = SB
   Case 4
     X1 = 0: Y1 = 0: X2 = 0: Y2 = 0
     XX1 = 0: YY1 = 0: XX2 = 0: YY2 = 0
End Select
End Sub
```

Case 3，5 To 12：这些代码段用 XYMove 方法来移动边界方框。当使用该公式时，必须在鼠标抬起事件中更新方框的位置。

Case 4：复位所有的值，因为它们可能与程序中其他代码冲突。

1.3.6 Movetest 程序小结

Movetest 程序还有一个 Blank 命令按钮，该按钮索引号为 8。你可能会发现，程序中有几个按钮专门用来在垂直或水平方向上缩放图形。实现这些功能的过程可以与其他的过程互换，你还可以交换 X，Y 变量、偏移值，等等，得到的结果却是相同的。该 Blank 命令按钮可用来插入一段混合的过程，以观察在屏幕上移动对象的结果。

1.4 本章小结

本章中应该牢记的重要内容是“边界方框”一词。本书中包括的几乎所有的程序都以某种方法使用了边界方框结构来移动、拉伸、翻转及调整桌面上的对象。

1. 边界方框是一个不可见的矩形，该矩形能够包住屏幕上的整个图形。
2. 边界方框的各条边通常用 left（左），top（上），right（右），bottom（下）或 W（西），N（北），E（东），S（南）或 X1，Y1，X2，Y2 来表示。

附注： 本书中主要用变量 W，N，E，S 来分别表示西，北，东，南。另外 X1，Y1，X2，Y2 被同一程序中的其他图形方法用作临时存储单元。Windows API 的 Rect 结构（将在第 8 章及第 15 章中讨论）使用的则是 left，top，right，bottom。

第 2 章　Artisan 程序

Artisan 程序是一个小型的绘图包，该软件包使用 Visual Basic 语言来创建各种圆、椭圆、方框、直线和贝齐尔曲线(Bézier curves)（图 2.1 给出了 Artisan 程序的一个屏幕画面）。设计 Artisan 时追求的目标就是在任何可能的情况下都使用 Basic 语句，避免使用 Windows API 函数。第 15 章中讲述的 ArtAPI 程序将本章中的大多数 Basic 代码过程转换成了 API 图形函数。部分代码在第 15 章中被重新改写以显示其简洁性（首先应该通过本章理解这些方法的基本原理）。

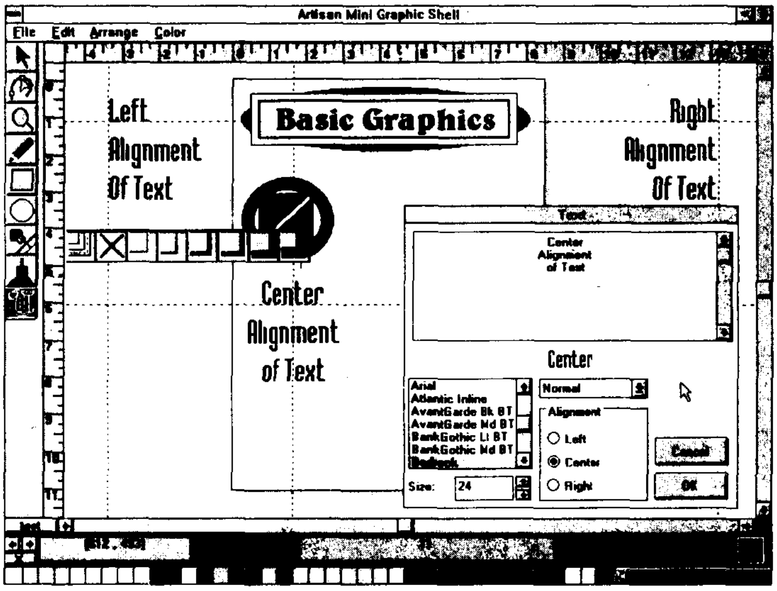

Artisan Mini Graphic Shell
File　Edit　Arrange　Color
Left Alignment Of Text
Basic Graphics
Right Alignment Of Text
Center Alignment of Text
Text
Center Alignment of Text
Center
Arial
Atlantic Inline
AvantGarde Bk BT
AvantGarde Md BT
BankGothic Lt BT
BankGothic Md BT
Size: 24
Normal
Alignment
○ Left
● Center
○ Right
Cancel
OK

图 2.1　Artisan 程序的屏幕画面

附注： API 是 Application Programming Interface（应用程序编程接口）的缩写。API 是一组函数，这些函数是 Windows 操作系统的一部分，可被其他的应用程序访问。因此 API 函数对于 Visual Basic 来说是外部的。

以下的图形功能不包括在 Artisan 程序中，因为它们在 Visual Basic 语言内并不实用。这些函数中有些（在以下的各节中列出）太长以至于本书容纳不下，还有一些则更适用于 Windows API 函数编写。

2.1　输入或输出过滤器

过滤器是指一些程序代码，这些代码能够解开其他商业产品图形文件的编码结构，大多数情况下，这些文件都是二进制文件，它们是在将图形文件存到磁盘上时生成的。因为这是一个充满竞争的世界，所以这样的信息是极少自由地与其他厂商共享的。唯一的方法就是自

已试着解开这些二进制编码结构，或者从生产厂家或第三方获得这些输入或输出代码的使用权。大多数软件公司以拥有大量的输入、输出过滤器而自豪，确实是这样。

在 Visual Basic 中，可以方便地输入并输出位图文件(.BMP)，Windows图元文件(.WMF)及文本文件(.TXT)，这都是标准的，在 Visual Basic 的参考指南中都有详细的解释。你也可以用 API 函数调用来处理(拉伸、旋转及裁剪)位图文件。

Visual Basic 中基于知识的帮助文件已经包括了这些图形过程的例子，因此没有必要再在本程序中包括进这些函数调用。另外，你可以购买第三方的自定义控制对象来输入和输出更多种类的图象。

2.2　页面设置、标尺和边界打印尺寸

页面设置尺寸是在 Update Desktop 过程中定义的，它是根据 800×600 的屏幕分辨率粗略地计算出来的。页就是用 Basic 的 Line 语句画出的一个矩形框。它的距离是从桌面的正中心开始度量的。若要改变页的大小，只需改变 Line 语句的起点和终点。将这些值换成一个变量，就可以按你自己的意愿来设置页大小。不过，如果你使用不同的屏幕驱动程序，页可能显示得不一样，这是因为所有的尺寸都是依据象素宽度而定的，不同的监视器(或不同的设备)，其象素宽度是不同的。在这种情况下，API 函数 GetDeviceCaps 总是用得着的，该函数在“启动时默认的窗体装载值”一节中解释。

2.3　所见即所得(WYSIWYG)

Windows 可以告诉你某个设备(监视器或打印机)每英寸能够显示多少象素以及它当前所用的分辨率。在你获得了这些信息后，就可以计算出适当的页大小、标尺尺寸、打印机图象边界及调整程序所需的恰当的缩放比例。

关键是在一个过程中计算出一个所见即所得(WYSIWYG)的表。Select/Case 语句就是一个很好的例子：用变量来改变刻度或比例属性。还应该考虑的是桌面或其他容器中所用的用户定义的缩放比例。本程序中所用的标尺就是这方面的一个很好的例子。因为标尺的标线和滚动过程都是根据 800×600 的屏幕分辨率设计的，将分辨率改为 640×480 就会破坏校准方式，改变主桌面区的 Left(左)，Top(上)，Width(宽)，Height(高)和 Scaling(比例)属性也是这样。

2.4　屏幕、打印机和分辨率

最后要考虑的事情就是打印出所画内容的能力，将打印机看作传送图象的第二监视器。打印机像一个次监视器一样，可以有不同的屏幕大小。因为桌面页的尺寸是从图片容器中央设置的，所以你也可以以同样的方式将图象传送至打印机。使用简单的数学方法就可以计算出打印机打印区域的中心位置，然后将图象置于中心位置。不过在大多数情况下，图象显示起来很小，这是因为大多数的打印机的图象区域像一个很大的监视器。补救方法是减去屏幕图形容器尺寸和打印机尺寸之间的差值。通过加大或者减小打印机的 top，left，width 或

height 尺寸，就可以将图象置于打印机页面的任何位置，也可以将图象调整到打印机可以处理的大小。

2.5 程序间的协调

设备定标(Scaling)和屏幕驱动程序分辨率在保持程序与自身及它所使用的任何其他设备同步方面起着重要的作用。鉴于此，最好是使用 Windows 的设备 API 函数来处理与其他硬件设备的兼容性问题。由于逻辑坐标和设备坐标之间的差别，对于某些人来说，使用 Windows API 函数可能会很混乱。本书后面的章节中使用了这样的函数，但是目前你还是可以用简单的数学方法来手工设置所有的打印机和屏幕位置。

Artisan 程序处于一个紧缩的窗体中，尽管功能有限，它却包含了 40 多个 Visual Basic 的对象和 34 个主要的过程。

声明段有若干数组结构，这些数组结构用来存储图象信息，随着功能的增加，这些结构会越来越大。由于上面所列的种种原因，本程序没有根据不同的屏幕分辨率来设置，而是根据 800×600 来设定的，否则，API 设备过程就会变得十分冗长。

附注： 因为 Artisan 程序太长，本章及随后的三章都用来详述该程序的各个细节。

2.6 在启动时自动重画(AutoRedraw)

在 Visual Basic 中，每个图片框都有两个位图图象与其相关联。第一个就是实际的屏幕图象，为了防止所画图形超出控制的边界区域或内部尺寸，该图象可能被裁剪。第二幅图象就是当控制对象的 AutoRedraw 属性设置成 true 时在内存中保存的图象。例如，假设在所画的图形上面显示出了一个对话框，则当该对话框隐去之后，Visual Basic 就应该恢复原来所画图形。Visual Basic 通过计算并在内存(持久)位图和控制对象的屏幕图象之间来回交换图象区域来作到这一点。

本程序中的主桌面绘图区是一个图片框控制对象，该控制对象叫作 Artwork。每当在 Artwork 控制对象内画了一个对象，或者将一个对象移至新位置时，该 Artwork 图片框图象就会被更新。正常情况下，对图片框控制对象内的图形图象的更新是通过该控制对象的 Paint 事件来处理的。但是，由于本程序中可以移动对象，因此在发生鼠标移动事件时会触发 Paint 事件，引起图象闪烁，降低性能。可以绕过 Artwork 的 Paint 事件，而在一个叫作 UpdateObjects 的过程中重画所有的图象。

每当调用 UpdateObjects 过程重画桌面页及对象时，Artwork 图片框的 AutoRedraw(自动重画)属性就被设成 True。为了滚动的目的，Artwork 图片框控制对象的大小被设置成接近监视器屏幕区域的两倍。AutoRedraw 保持了一个内存(持久)位图，该位图包括了整个图片框区域，这样，滚动时就可以替换掉被裁剪的图象。存储 Artwork 图片的位图图象需要 1.5MB 的内存。裁剪持久位图图象以节省内存是不可能的，这是因为 AutoRedraw 采用一种 in-house 功能，该功能将整个控制对象的内部尺寸定义成内存位图。

注意： 如果看到一条 Cannot Create AutoRedraw(不能创建自动重画)消息，说明没有

所需的 1.5MB 内存，必须将 Artwork. AutoRedraw 语句注释掉(加上')，以获得更多的内存，或者改变 Artwork 图片框的大小，以适合屏幕的尺寸。如果采用了后一种方法，则标尺和滚动条比例过程必须重新改写。在改变大小及改变比例前，请先使用 Scaltest 程序(参见第 14 章)找到合适的方法。

2.7 学习过程

Artisan 程序是一个绘图程序，该程序使用了几乎所有的 Visual Basic 语句，并有意地在任何可能的时候都避免使用 Windows API 函数。尽管它功能很强，但仍有一定的局限性。此处只是用 Artisan 做做试验，然后再用第 14 章“Scaling 和 Printing 程序”中的 Scaltest 程序作为外壳来建立自己的图形软件包。当理解了绘图包的基本原理后，就可以加进本书所提供的各种程序中的过程，并可以将第 8 章“API 绘图程序”给出的 ApiDraw 程序中 API 绘图代码或者第 15 章“ArtAPI”给出的程序，以及 Artisan 程序中的 Basic 语言代码组合起来设计自己的程序。

为了减少过程和数组的长度，本程序没有将对象分组。第 13 章解释了怎样实现这一点，该章讨论了节点连接程序；第 15 章也阐述了这一方法，它讨论的是本章程序的 API 版本。

Zoom 工具仅支持两种 Zoom 模式，原因是所有文字都是用 Print 语句输出到桌面上的。本书后面将解释为什么用 Visual Basic 的 print 语句来实现 WYSIWYG(所见即所得)的文字种类是有限的。稍后，你还会看到怎样用 Scaltest 程序来精细调整控制对象的比例及缩放值，以满足你的需要。最后，如果监视器或打印机对正确的图形对齐方式没有反应，那么可能是比例和监视器分辨率发生了冲突。第 14 章中讨论的 Scaltest 程序使用了与 Artisan 程序中相同的 Visual Basic 对象名和比例属性。在修改 Artisan 程序之前，应该仔细调整 scaltest 程序以满足自己的需要。

2.8 程序速度和图片框

画出一个对象后，在每条线或每个曲线段的起点和终点就会出现白色的小方框，正是这些白色小方框构成了该对象的形状。这些白色小方框叫做节点，它们被用来编辑一个形状的轮廓。不过本程序只能将节点随直线和曲线对象一起移动。有一种曲线对象叫做 Bézier 曲线，在这种曲线起点和终点之间还有一些折线，这些折线的两端有黑色的小方框，叫做图柄，它们被用来调整曲线的外形。在第 15 章的 ArtAPI 程序中将进一步介绍如何利用对象节点和图柄图片框控制对象来编辑复杂的多边形。

也许最引人注目的引起性能降低的事件就是重新确定加亮显示的对象周围各个图柄及节点的位置，本程序中所用的全部图柄和节点都是 Visual Basic 的图片框，使用这种类型的控制对象省去了许多为鼠标事件编写的代码。每个控制对象都必须被设成不可见的，将其移走并重新设置比例，每当选定了对象时再将它们设成可见的。如果缩放工具从程序中删去了，则图片框的重设比例功能也可删去。只有窗体、图片框和打印机对象能使用 Visual Basic 的 Scale(设定比例)方法。第 15 章中讲述的 ArtAPI 程序将许多图片框控制对象换成了图象控制对象，图象控制对象属性少一些，因此显示起来更快些。另外，还应该加上一些代码，以

调整图象控制对象的内部鼠标移动，使其能与 Artwork 图片框的比例相匹配。

另一种方法是仿真一个能对鼠标事件作出响应的控制对象。在 Tabs 程序中可以找到这样的例子，该程序位于随书磁盘的 16MISC 目录下。

2.9 工具按钮图片框

各种工具按钮构成了一个工具箱的图象，这个工具箱显示在一个图片框控制对象中。工具用来绘制一个形状，或者用来执行一项诸如填充或改变一个形状的边框宽度之类的任务。例如，若要画一个方框，就应该选定其上有一个方框图标的工具，然后用鼠标在桌面上定位，准备开始画方框，单击鼠标左键，按住鼠标键不放，拖动鼠标，就会显示出一个方框形状。放开鼠标键后，方框就完成了。也可以用一个看上去像一把画刷图标的按钮来将一个方框填充成某种颜色。

在随本书一起提供的附带磁盘的 BITMAPS 目录下可以找到位图工具箱图象。可以用任意一种 BMP 绘图软件或 Microsoft Windows 提供的 Paint 程序来编辑每一个工具的外观。下面从顶层工具按钮开始说明每一个工具的功能。

屏幕左侧的 ToolButtons 控制对象有以下功能：

- pointer tool（指针工具）用来选定并高亮显示对象，或将对象移至屏幕上别的地方。可以用指针工具画一个包围住所要选定的对象的方框（拖动鼠标就会出现一个虚线框）。不过，本程序不能选定多个对象。也可以用指针工具拉伸一个对象，或拉动标尺上的标线。当选定了其他工具时，按空格键就会触发指针工具。本程序不能拉伸对象，若想做到这一点，最好是使用一种自定义的字体处理程序，这种字体处理程序提取出字体 GlyphOutline 轮廓（构成每个字符的多边形或 Bézier 轮廓）以供文字处理之用。标准的 API Text 函数可以使字体变宽、变窄、变厚、变薄、旋转，但是这些函数不能对字体图象作镜象或反转变换。
- Bézier tool（Bézier 工具）用来将直线转换成 Bézier 曲线，或者反过来。也可以用该工具拖曳直线的一个端点以移动该直线的另一个端点。此工具是绘图程序的灵魂部分和核心部分，并可用来操纵任何对象的节点。但是，Bézier 工具并不移动方框、圆或文字对象的节点。而且，本程序也没有连接或组合对象。第 13 章“节点连接程序”将涉及此论题，并展示怎样操纵复杂对象。

附注： 节点（node point）是白色小方框连接对象的终止或起始坐标的地方。

- zoom tool（缩放工具）用来按不同的比例查看页和对象。第 14 章给出了几个使用这种工具的例子，这些工具有多种缩放比例。当在多个缩放范围内缩放文字时，Visual Basic 通常将文字点大小调整至原来的四分之一，这与图形的 WYSIWYG（所见即所得）方法是相互冲突的。大多数绘图程序都按自定义 .DLL 文件的方式使用自己的字体生成机制。用 Windows 的 GyphOutline API 函数来创建字体机制是可能的，但这需要有高级字体、B 样条曲线及内存管理方面的技术。不过本书并未涉及字体。
- line tool（直线工具）用来绘制直线。你也许想用 API 函数绘制直线——或者任何其他对象。第 8 章和第 13 章给出了 API 绘图函数的例子。本例中，API 函数 LineTo 用

来绘制曲线或直线，效果很好。

- box tool（方框工具）用来绘制矩形方框或正方框。如果想要方框的角是圆角，请参考第 11 章“圆角程序”或第 8 章，以获取细节知识。
- circle tool（圆工具）用来绘制圆或椭圆对象。API 函数 Ellipse 用来完成此项任务是最合适不过了。这个 API 函数已经包括了正圆函数，参阅第 8 章可以找到该函数的一个很好的例子。
- outline tool（外框工具）用来改变一个对象外框的颜色，或删去对象的外框。
- fill tool（paint tool）（填充工具）在本程序中用来填充一个圆或方框对象的内部。这正是 Basic 语言需要使用许多图形语句来处理复杂对象的地方。

  Basic 中原有的 Paint 语句已被 Microsoft 弃置不用，最简便的使用填充的方法是使用 Windows 的 API 函数 FloodFill 或 API 函数 PolyFillMode 和 Polygon。Windows API 库中还有许多矩形和椭圆形填充的函数（有关 API 图形函数的更多的信息请参见第 8 章）。
- text tool（文字工具）用来在页面或桌面上显示文字。如前所述，Visual Basic 打印文字时是有局限性的。第 12 章“文字对齐程序”讲述的 APItext 程序代码可以在修改之后用来改变文字的高度、宽度和旋转角度。不过 APItext 程序也有局限性。对于真正的文字处理，还需要将字体的 GyphOutlines 轮廓转换为多个片段，这是一种高级的字体技术，本书不涉及这种技术。

2.10 颜色栏调色板

颜色栏（ColorBar）调色板是一个图片框，该图片框显示了若干种颜色。该调色板位于屏幕的底部。使用调色板左边的向左和向右滚动按钮可以显示余下的颜色。滚动时，出现的每一种新颜色都通过图片框 ColorBar 的 Paint 事件处理。这个例子是通过修改 Visual Basic 提供的 IconWorks 例子得来的。将颜色更新放在 Paint 事件中（与 Artisan 程序中一样）降低了调色板的滚动速度。加快调色板滚动速度的替代方法请参见第 15 章。

颜色栏（color bar）图片框控制对象涉及以下事件：

- 选定了一个方框或圆对象后，可以用鼠标左键单击调色板上的某种颜色，以将该对象填充成此颜色。
- 选定了一个方框、圆或直线对象后，可以用鼠标右键单击调色板上的某种颜色，将该对象的边框改成此颜色。
- color scroll（颜色滚动）控制对象位于 Test 按钮的下方，用于控制 ColorBar 调色板的左、右滚动，如图 2.2 所示。该调色板中所用的颜色值与 Visual Basic 提供的 IconWorks 例子中所记录的颜色值相同。如果要改变这些值，请参考第 4 章中的 ColorBar 过程，还必须适当修改 ARTISAN7.BAS 文件（参见附录 A）。
- 按 no fill（不填充）（用字符“X”标记）按钮就会擦去当前选定的方框或圆对象所填充的颜色，该按钮紧挨在工具控制对象的下方。
- status color（状态颜色）图片框控制显示对象的当前边框颜色，该控制对象位于屏幕的右下部。紧挨着它的是 status paint（填充状态）图片框，用于显示对象的当前填充

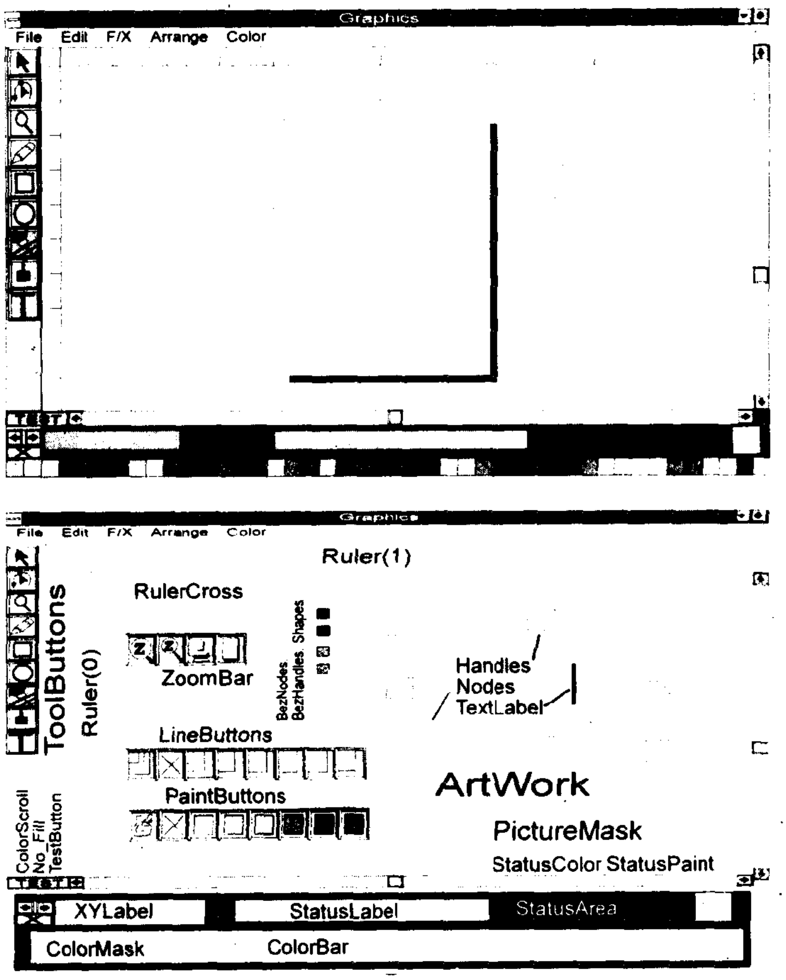

Graphics
File
Edit
F/X
Arrange
Color
Ruler(1)
RulerCross
ToolButtons
Ruler(0)
ZoomBar
BezNodes
BezHandles
Shapes
Handles
Nodes
TextLabel
LineButtons
PaintButtons
ArtWork
PictureMask
StatusColor StatusPaint
ColorScroll
No_Fill
TestButton
XYLabel
StatusLabel
StatusArea
ColorMask
ColorBar

图 2.2　Artisan 主窗体及其控制对象

颜色。

- test(测试)按钮控制对象可用来测试代码。

2.11　滚动条控制对象

滚动条控制对象是 Visual Basic 的标准水平和垂直滚动控制对象。它们用来在 Artwork 图片框控制对象的显示区域内左、右、上、下移动绘制的图象。

HScroll 和 VScroll 控制对象有以下事件：

- HScroll 水平移动桌面。
- VScroll 垂直移动桌面。在 Zoom In 与 Zoom Out 事件中，并没有将其调整到 Maximum(最大值)和 Minimum(最小值)。可以用第 14 章中的 Scaltest 程序对显示的图象进行细致地调整。

2.12　Status Area（状态区域）框

Status Area（状态区域）显示以下信息：

- XYLabel 控制对象显示鼠标指针在 ArtWork 区域中的当前 X，Y 位置，以及该区域中控制对象内部的 X，Y 坐标（例如，图柄、节点、Bézier 节点和 Bézier 图柄）。显示的所有值都是按 twip 度量，然后再转换为象素值。
- StatusLabel 控制对象显示要告知用户的多种信息。对该控制对象的标准调用方式是 StatusLabel. Caption＝Format $（变量）。StatusLabel 可以显示当前对象索引号、拉伸边界方框的 twip 度量值，或者用字体对话框选定的当前字体。

2.13　菜单区域

菜单区域是一个标准的 Windows 下拉式菜单栏，在该菜单栏中，可以打开、保存及删除所绘制的对象。菜单区域功能有限，这是因为要获得特殊的效果或其他功能就必须增加代码。Artisan 程序包含了一些非常有用的在自己的程序中打开、保存或打印图形的替代方法。大多数代码采用了很长的公式。奇怪的循环、复杂的数组结构被去掉，因此即使是初学者也很容易理解其基本原理。例如 Edit 菜单中的删除对象选项清除了 30 个属性值，任何绘制的形状都有这 30 个属性值，编码如清单 2.1 所示。不过，为了清除每个属性值用了好几行代码。随着你继续阅读本书，你会明显地发现，这些属性值都可以很方便地存进临时变量中，再插入一些代码，这些变量就可以用来剪切和粘贴对象。

清单 2.1　从桌面上删除一个对象

```
Sub Menu_EditSelection_Click (Index As Integer)
Select Case Index
    Case 0 'UNDO
    ''''''''''''''''
    Case 1 'REDO
    ''''''''''''''''
    Case 3 'CUT
    ''''''''''''''''
    Case 4 'COPY
    ''''''''''''''''
    Case 5 'PASTE
    ''''''''''''''''
    Case 6 'DELETE
If B = 0 Then Exit Sub
If Node(0).Visible = True Then
N_H_Clear
FillValue(B) = 0; LineValue(B) = 0; OutLineColor(B) = 0
PaintColor(B) = 0; LineWidth(B) = 1; FlagAlign(B) = 0
OFFSET(B) = 0; TextSize(B) = 0; TextFont(B) = ""
TextBold(B) = 0; TextItalic(B) = 0; CarriageCount(B) = 0
Storedtext(B) = ""
```

```
FlagBézier(B) = 0: FlagLine(B) = 0: FlagBox(B) = 0
FlagCircle(B) = 0: FlagText(B) = 0
P1X(B) = 0: P1Y(B) = 0: P2X(B) = 0: p2Y(B) = 0
h1X(B) = 0: h1Y(B) = 0: h2X(B) = 0: h2Y(B) = 0
DeleteObject = True
XX1 = W(B): YY1 = N(B): XX2 = E(B): YY2 = S(B)
W(B) = 0: N(B) = 0: E(B) = 0: S(B) = 0
    UpdateObjects
    DeleteObject = False
End If
End Select
End Sub
```

第 5 章中 Artisan 程序的 To_Front_Back 过程请参见清单 2.2。

清单 2.2 To_Front_Back 过程

```
Sub To_Front_Back (Index As Integer)
Dim PC As Long: Dim OLC As Long
Dim St As String: Dim TF As String
If Node(0).Visible = False Then Exit Sub
If Index = 0 And B = Total Then Exit Sub
If Index = 1 And B <= 1 Then Exit Sub
''''''store objects attributes
If FlagText(B) = B Then FT = 1
If FlagCircle(B) = B Then FC = 1
If FlagLine(B) = B Then FL = 1
If FlagBox(B) = B Then FB = 1
If FlagBézier(B) = B Then Bez = 1
OLC = OutLineColor(B): PC = PaintColor(B): LW = LineWidth(B)
Fill = FillValue(B): LV = LineValue(B): St = Storedtext(B)
TS = TextSize(B): TF = TextFont(B): TB = TextBold(B): TI = TextItalic(B)
FA = FlagAlign(B): OF = OFFSET(B): CR = CarriageCount(B)
W1 = W(B): N2 = N(B): E3 = E(B): S4 = S(B)
p1XX = P1X(B): p1YY = P1Y(B): p2XX = P2X(B): p2YY = p2Y(B)
h1XX = h1X(B): h1YY = h1Y(B): h2XX = h2X(B): h2YY = h2Y(B)
'''''''End of every possible attribute
If Index = 0 Then Last = Total - 1: First = 1 Else Last = 2: First = -1
For B = B To Last Step First: A = B + (First)
W(B) = W(A): N(B) = N(A): E(B) = E(A): S(B) = S(A)
P1X(B) = P1X(A): P1Y(B) = P1Y(A)
P2X(B) = P2X(A): p2Y(B) = p2Y(A)
h1X(B) = h1X(A): h1Y(B) = h1Y(A)
h2X(B) = h2X(A): h2Y(B) = h2Y(A)
OutLineColor(B) = OutLineColor(A)
PaintColor(B) = PaintColor(A)
LineWidth(B) = LineWidth(A)
FillValue(B) = FillValue(A)
LineValue(B) = LineValue(A)
Storedtext(B) = Storedtext(A)
TextSize(B) = TextSize(A)
TextFont(B) = TextFont(A)
TextBold(B) = TextBold(A)
```

```
TextItalic(B) = TextItalic(A)
FlagAlign(B) = FlagAlign(A)
OFFSET(B) = OFFSET(A)
CarriageCount(B) = CarriageCount(A)
If FlagText(A) = 0 Then FlagText(B) = 0 Else FlagText(B) = B
If FlagCircle(A) = 0 Then FlagCircle(B) = 0 Else FlagCircle(B) = B
If FlagLine(A) = 0 Then FlagLine(B) = 0 Else FlagLine(B) = B
If FlagBox(A) = 0 Then FlagBox(B) = 0 Else FlagBox(B) = B
If FlagBézier(A) = 0 Then FlagBézier(B) = 0 Else FlagBézier(B) = B
Next B

''''Reset objects attributes
If FT = 1 Then FlagText(B) = B Else FlagText(B) = 0
If FB = 1 Then FlagBox(B) = B Else FlagBox(B) = 0
If FC = 1 Then FlagCircle(B) = B Else FlagCircle(B) = 0
If FL = 1 Then FlagLine(B) = B Else FlagLine(B) = 0
If Bez = 1 Then FlagBézier(B) = B Else FlagBézier(B) = 0
OutLineColor(B) = OLC: PaintColor(B) = PC: LineWidth(B) = LW
FillValue(B) = Fill: LineValue(B) = LV: Storedtext(B) = St
TextSize(B) = TS: TextFont(B) = TF: TextItalic(B) = TB: TextItalic(B) = TI
FlagAlign(B) = FA: OFFSET(B) = OF
CarriageCount(B) = CR
W(B) = W1: N(B) = N2: E(B) = E3: S(B) = S4
P1X(B) = p1XX: P1Y(B) = p1YY: P2X(B) = p2XX: p2Y(B) = p2YY
h1X(B) = h1XX: h1Y(B) = h1YY: h2X(B) = h2XX: h2Y(B) = h2YY
UpdateObjects
End Sub
```

第15章中 ArtAPI 程序的 To_Front_Back 过程请参见清单2.3。

清单 2.3　To_Front_Back 过程

```
Sub To_Front_Back(Index As Integer)
    If B = 0 Then Beep:Exit Sub
    If PolyShape(B) = B Then B = FirstPolyPt(B)

    Select Case Index
    Case 0 'To Front
    For Objectnumber = 1 To Total
        Z = DrawOrder(Objectnumber)
        If Z = B Then Exit For
    Next Objectnumber 'Objectnumber will highlighted object
  Temp = DrawOrder(Objectnumber)

For Repeat2 = Objectnumber To Total
    NextA = Repeate2 + 1
    DrawOrder(Repeat2) = DrawOrder(NextA)
Next Repeat2
DrawOrder(Total) = Temp

Case 1 'To back
    For Objectnumber = 1 To Total
        Z = DrawOrder(Objectnumber)
        If Z = B Then Exit For
    Next Objectnumber 'Objectnumber will highlighted object
  Temp = DrawOrder(Objectnumber)
    For Repeat2 = Objectnumber To Total
    NextA = Repeate2 - 1
    DrawOrder(Repeat2) = DrawOrder(NextA)
    Next Repeat2
  DrawOrder(Total) = Temp
  End Select
            UpdateObjects
End Sub
```

2.13.1 File 菜单

File 菜单是 Artisan 程序的第一个菜单项。单击 File 菜单就会显示出下列选项：

- New 菜单选项清除所有变量数组并开始绘制新图。
- 在本程序的 Visual Basic 3.0 和 4.0 版中，Open 菜单选项显示出一个打开文件对话框，使得用户可以打开一个已存在的图形文件，默认的文件扩展名是 .ART。
- 在本程序的 Visual Basic 3.0 版及更高版本中，Save 和 Save As 菜单选项打开一个保存文件对话框，使得用户可以保存当前所画的图形。
- Exit 菜单选项退出该程序，但是当前所绘制的图形不会自动保存。

2.13.2 Edit 菜单

Edit 菜单是 Artisan 程序中的第二个菜单项。单击 Edit 菜单就会显示出以下选项：

- Delete 菜单选项和 Delete 键可擦去当前选定的对象。

2.13.3 Arrange 菜单

Arrange 菜单是 Artisan 程序中的第三个菜单项。单击 Arrange 菜单就会显示出下列选项：

- To Front 菜单选项将选定的对象移至桌面上当前显示的所有对象之上。
- To Back 菜单选项将当前选定的对象移至当前显示的对象之下。

2.13.4 Color 菜单

Color 菜单是 Artisan 程序中的第四个菜单项。单击 Color 菜单就会显示出下列选项：

- Show Rulers 菜单隐藏或显示标尺。

2.13.5 Color Palette 子菜单

Color Palette 子菜单是 Artisan 程序中 Color 菜单的第二个菜单项，单击 Color Palette 子菜单就会显示下列选项：

- Edit Palette 菜单选项向当前调色板中加进一种新颜色。用 RGB 滚动条选定一种颜色并将颜色从大的图片框（该图片框显示新颜色）中拖出，光标就会变成一个 Felt

Marker 图标。将该图标放在任一颜色行中，新颜色就会出现在颜色图及 ColorBar 调色板中。

- Default Palette 菜单选项装入 ARTISAN7. BAS 文件定义的默认调色板。
- Custom Palette 菜单选项装入自定义的调色板，该调色板存储在 ARTISAN. INI 文件中，只有当保存自定义颜色时才会生成此文件。
- Save As Custom 菜单选项将新调色板中的颜色值写入 ARTISAN. INI 文件中，以后就可以装入该调色板。需要的时候可以删去或覆盖该文件，或者使用默认调色板。

2.14　鼠标器事件

鼠标器事件就是用鼠标器执行的一些动作。在本程序中，鼠标左键按下、双击及右键按下事件可以触发下列程序过程：

- 鼠标左键可用来选定对象、移动对象、围绕一个对象画一个方框以选定该对象、移动一条直线或 Bézier 曲线的节点、画一个方框、圆或直线，拖动标尺线以及选定一种填充颜色。
- 当选定了一条直线或曲线之后，就可以双击鼠标左键显示 Edit Node（编辑节点）窗体，同时 Bézier(Shape)工具也必须被选定。接下来，就可以用 Edit Node 的选项按钮将直线转换成 Bézier 曲线。
- 当选定了 text（文字）工具时，鼠标右键可用来打开一个功能有限的文字编辑器；还可以显示上一次选定菜单的子菜单项，并可选定对象的边框颜色。

Artisan 程序的窗体和控制对象的属性在表 2.1 中列出。

表 2.1　Artisan 程序窗体及控制对象的属性

Object：Form
Object Name：Artisan

BackColor　= &H00404040&
ForeColor　= &H00FFFFFF&
Left　= 165
Top　= 1065
Caption　= "Artisan Mini Graphic Shell"
Height　= 7095
ScaleMode　= 3' Pixedl
Width　= 7260

Object：CommonDialog
Object Name：CMDialog1

Left　= 0
Top　= 4410

Object：Picture Box
Object Name：RulerCross

BackColor　= &H0000FFFF&
Left　= 540
Width　= 345
Height　= 345
Top　= 0

Object：Picture Box
Object Name：StatusArea

Align　= 2' Align Bottom
BorderStyle　= 0' None
BackColor　= &H00000000&
ForeColor　= &H00FFFFFF&

```
    Height      =1455                Left         =0
    Top         =4980                Width        =7170
```

Object:Picture Box

Object Name:ColorMask

```
    Height      =435                 Left         =within StatusArea
    ScaleHeight =27                  ScaleMode    =3' Pixel
    Top         =within StatusArea   Width        =7155
```

Object:Picture Box

Object Name:ColorBar

```
    Height      =270                 Left         =within ColorMask
    ScaleHeight =1                   ScaleMode    =0' User
    Top         =within ColorMask    Width        =6930
```

Object:Picture Box

Object Name:StatusColor

```
    BorderStyle =0' None             Hehight      =390
    Left        =within StatusArea   ScaleMode    =3' Pixel
    Top         =within StatusArea   Width        =450
```

Object:Picture Box

Object Name:StatusPaint

```
    BorderStyle =0' None             Height       =315
    Left        =within StatusColor  Top          =within StatusColor
    Width       =345
```

Object:HScrollBar

Object Name:ColorScroll

```
    Height      =255                 LargeChange  =80
    Left        =within StatusArea   Max          =2000
    SmallChange =16                  Top          =within StatusArea
    Width       =510
```

Object:CommandButton

Object Name:No_Fill

```
    Caption     ="X"                 Height       =195
    Left        =within StatusArea   Top          =within StatusArea
    Width       =495
```

Object:HScrollBar

Object Name:HScroll

```
    Height      =240                 LargeChange  =100
    Left        =810                 Max          =200
    Min         =-20                 SmallChnage  =10
    Top         =45                  Width        =5985
```

Object:CommandButton

Object Name:TestButton

```
    Caption     ="test"              Height       =285
    Left        =within StatusArea   Top          =within StatusArea
    Width       =780
```

Object:Label

Object Name:XYLabel

```
    Alignment   =2' Center           BackColor    =&H00FFFF00&
```

BorderStyle =1' Fixed Single　Height =390
Left =within StatusArea　Top =within StatusArea
Width =1125

Object:Label
Object Name:StatusLabel

Alignment =2' Center　BackColor =&H00FFFF00&
BorderStyle =1' Fixed Single　Height =390
Left =within StatusArea　Top =within StatusArea
Width =2775

Object:Picture Box
Object Name:Ruler

BackColor =&H00C0C0C0&　Height =4335
Index =0　Left =540
Top =405　Width =330

Object:VScrollBar
Object Name:VScroll

Height =4725　LargeChange =100
Left =6930　Max =250
Min =−250　SmallChnage =10
Top =0　Width =255

Object:Picture Box
Object Name:Ruler

BackColor =&H00C0C0C0&　Height =345
Index =1　Left =900
ScaleHeight =315　ScaleWidth =5985
Top =0　Width =6015

Object:Picture Box
Object Name:PaintButtons

Height =510　Left =1035
Picture =Paint. BMP　ScaleMode =3' Pixel
Top =3960　Visible =0' False
Width =3840

Object:Picture Box
Object Name:ZoomBar

Height =510　Left =1020
Picture =Zoom. BMP　ScaleMode =3' Pixel
Top =1020　Visible =0' False
Width =1920

Object:Picture Box
Object Name:LineButtons

Height =510　Left =1035
Picture =Linetool. BMP　ScaleMode =3' Pixel
Visible =0' Pixel　Width =3840

Object:Picture Box
Object Name:PictureMask

BorderStyle =0' None　Height =4245
Left =945　ScaleMode =3' Pixel
Top =495　Width =5865

Object:Picture Box

Object Name:ArtWork

```
BackColor      =&H00FFFFFF&          BorderStyle    =0' None
ClipControls   =0' False             FontName       =" Arial"
FontSize       =12                   Hight          =2325
Left           =within PictureMask   Height         =within PictureMask
Width          =3570
```

Object:Picture Box

Object Name:ScaleBox

```
Height         =330                  Left           =within Artwork
Top            =within Artwork       Visible        =0' False
Width          =330
```

Object:Picture Box

Object Name:Handle 8 indexed controls

```
BackColor      =&H000000FF&          BorderStyle    =0' None
Height         =120                  Index          =0 through 7
Left           =within Artwork       MousePointer   =2' Cross
ScaleHeight    =120                  ScaleWidth     =120
Top            =within Artwork       Visible        =0' False
Width          =120
```

Object:Picture Box

Object Name:BezNode2

```
BackColor      =&H0000C000&          BorderStyle    =0' None
Height         =105                  Left           =with Artwork
ScaleHeight    =105                  ScaleWidth     =105
Top            =within Artwork       Visible        =0' False
Width          =105
```

Object:Picture Box

Object Name:BezNode1

```
BackColor      =&H0000C000&          BorderStyle    =0' None
Height         =105                  Left           =within Artwork
ScaleHeight    =105                  ScaleWidth     =105
Top            =within Artwork       Visible        =0' False
Width          =105
```

Object:Picture Box

Object Name:BezHandle2

```
BackColor      =&H00FF00FF&          BorderStyle    =0' None
Height         =105                  Left           =within Artwork
ScaleHeight    =105                  ScaleWidth     =105
Top            =within Artwork       Visible        =0' False
Width          =105
```

Object:Picture Box

Object Name:BezHandle1

```
BackColor      =&H00FF00FF&          BorderStyle    =0' None
Height         =105                  Left           =within Artwork
ScaleHeight    =105                  ScaleWidth     =105
Top            =within Artwork       Visible        =0' False
Width          =105
```

Object:Picture Box

Object Name:Node 4 indexed controls

```
    BackColor      =&H00000000&       BorderStyle    =0' None
    Height         =90                Index          =0 through 3
    Left           =within Artwork    ScaleHeight    =90
    ScaleWidth     =90                Top            =within Artwork
    Visible        =0' False          Width          =90
```

Object:Label

Object Name:Txt

```
    AutoSize       =-1 'True          BackColor      =&H00FFFFFF&
    BackStyle      =0' Transparent    BorderStyle    =1' Fixed Single
    Height         =480               Left           =within Artwork
    Top            =within Artwork    Visible        =0' False
    Width          =15
```

Object:Shape

Object Name:Shape2

```
    BackColor      =&H00FF0000&       BackStyle      =1' Opaque
    BorderStyle    =0' Transparent    Height         =105
    Left           =within Artwork    Top            =within Artwork
    Visible        =0' False          Width          =105
```

Object:Shape

Object Name:Shape1

```
    BackColor      =&H00FF0000&       BackStyle      =1' Opaque
    BorderStyle    =0' Transparent    Height         =105
    Left           =within Artwork    Top            =within Artwork
    Visible        =0' False          Width          =105
```

Object:Picture Box

Object Name:ToolButtons

```
    AutoSize       =-1 'True          BackColor      =&H00C0C0C0&
    BorderStyle    =0' None           DrawMode       =6' Invert
    Height         =4320              Left           =0
    Picture        =Toolbar.BMP       ScaleHeight    =288
    ScaleMode      =3' Pixel          ScaleWidth     =36
    Top            =45                Width          =540
```

Object:Menu

Object Name:Menu_File

```
                   ="&File"
```

Menu_FileSelection

```
    Caption        ="&New"            Index          =0
```

Menu_FileSelection

```
    Caption        ="&Open..."        Index          =1
```

Menu_FileSelection

```
    Caption        ="&Save"           Index          =2
    Shortcut       =+{F12}
```

Menu_FileSelection

```
    Caption        ="Save &As..."     Index          =3
    Shortcut       ={F12}
```

```
Menu_FileSelection
     Caption       ="&Print"           Index       =4
Menu_FileSelection
     Caption       ="E&xit"            Index       =5
Menu_Edit
     Caption       ="&Edit"
Menu_EditSelection
     Caption       ="&Delete"          Index       =6
Menu_Arrange
     Caption       ="&Arrange"
Menu_ArrangeSelection
     Caption       ="To &Front"        Index       =0
Menu_ArrangeSelection
     Caption       "To &Back"          =Index      =1
Menu_ArrangeSelection
     Caption       ="POPUP MENU"       Checked     =-1'True
     Index         =3
Menu_Color
     Caption       ="&Color"
Menu_ColorSelection
     Caption       ="&Color Palette"   Index       =0
Menu_Palette
     Caption       ="&Edit Palette"    Index       =0
Menu_Palette
     Caption       ="&Default Palette" Index       =1
Menu_Palette
     Caption       ="&Custom Palette"  Index       =2
Menu_Palette
     Caption       ="Save As Custom"   Index       =3
Menu_ColorSelection
     Caption       ="-"                Index       =1
Menu_ColorSelection
     Caption       ="Show &Rulers"     Checked     =-1'True
     Index         =2
```

2.15 定义变量、数组和结构

下面的例子定义了一个有三个元素的数组：

```
Dim Varname (1 To 3)%
```

数组是存储在内存中表的一种表示方式，可以把它看作是一张带有索引的图表。通过声明语句 Dim 将信息存储在该图表中，Dim 后应跟有一个任意的字母或单词，该字母或单词

被选来代表相应的主题。该图表的标题就是此变量。

在变量名之后有一对圆括号(),圆括号中是数组索引号。每一个数组项也是一个可以保存信息的变量,可以将每个数组元素看作是图表中的一个单元。

在要求更大的单元之前,一个单个的单元只能保存数量有限的信息。因此应该告知变量需要多大的单元格来保存信息。单元格的大小就是数据类型。单元格中可以使用的最小数据类型是 integer(整数)。下面是一个例子:

```
Dim CH As Integer
```

此例中,CH 可以存储一个任意大小的整数值,只要它在整数范围之内。Visual Basic 使用的有符号整数范围是－32768～32767。

下面一行代码将一个值存入变量 CH 中:

```
CH=27425+5343
```

但是,你不能存储下面的值,因为它超出了整数的范围:

```
CH=27425+5343
```

一个字节(byte)可以保存一个不大于 255 的值。一个整数使用 2 个字节,这两个字节分别叫作低字节和高字节。例如,低字节+(256×高字节)=总值,或者:

```
255+(256×255)=65535
-32768 to 32767=65535
```

如果超出了某种数据类型的范围,Visual Basic 就会产生一条溢出错误。这时就必须将此变量存在一个更大的单元中。比较大的数据类型有 long(长整数),single(单精度),double(双精度),currency(货币),string(串)以及用户定义类型。

2.15.1　数据类型

可以在每个 Dim 语句后使用数据类型的简略形式,而不必写其全名(例如 integer)。

数据类型	大小
Integer %(整数)	2 字节
Long&(长整数)	4 字节
Single!(单精度)	4 字节
Double #(双精度)	8 字节
Currency@(货币)	8 字节
String 4 bytes $(串)	* 1 字节

* 每个字符占一个字节,字符串的最大长度约为 65535(通常不超过 63000)

有关上面所列各种数据类型的值的范围请参考 Visual Basic 语言参考手册中的数据类型部分。

简言之,整数使用 integer 和 long,十进制小数用 single 和 double,文字用 string,而货币则用 currency。

索引数组是带索引图表的扩展，它意味着在一个变量名下定义多个单元块：

```
Dim Txt(1 to 200) As String
```

本例中，可以在一个标题叫做 Txt 的图表中存储多达 200 个文字项，(1 to 200)定义了每个文字串的索引号。

BoundingBox_MouseUp 过程使用这种类型的索引数组存储了几乎每个绘制对象的属性。

清单 2.4 BoundingBox_Up 过程

```
Sub BoundingBox_Up (X1, Y1, X2, Y2)
If MoveMouse() = True Then
  Total = Total + 1: B = Total
  StatusLabel.Caption = Format $ (B)
  LineWidth(B) = Artwork.DrawWidth
  W(B) = X1: E(B) = X2: N(B) = Y1: S(B) = Y2
    Select Case CurrentTool
      Case PointerTool
        Exit Sub
      Case BézierTool
        Exit Sub
      Case ZoomTool
        Exit Sub
      Case LineTool
        FlagLine(B) = B
        Nodes
      Case BoxTool
        FlagBox(B) = B
        FillValue(B) = 1
        Nodes
      Case CircleTool
        FlagCircle(B) = B
        FillValue(B) = 1
        Nodes
      Case TextTool
        'no code needed here
  End Select
End If
End Sub
```

Artisan 程序声明的大部分变量都采用有限(Limit)作为索引数组的元素个数，Limit 表示 1000 个元素，最后一个索引是 999，因为第一个单元的计数为 0。

2.15.2 用户定义的数据类型变量

用户定义型变量与数组变量类似，不同的是它们使用用户定义的子标题(下标)来检索或存储数据。你可以用多个子标题来定义数据。如果不使用用户定义的 Type 结构，大多数 API 图形函数都不能正常工作。这些数据类型可以指向变量名并从一个列表中抽取出值。下面是 Type 结构的一个例子：

```
Type PointAPI
    X As Integer
    Y As Integer
End Type
```

通常，如果按 Dim DT As Integer 形式声明一个变量，则 DT 就可以存储任意一个整数值。如果将该变量的数据类型从 Integer 改为 PointAPI，也就是改为 Dim DT As PointAPI，则 DT 就可以存储两个值。本例中，两个存放数值的单元叫做 X 和 Y，它们被声明为 integer（整数）。

现在就可以像下面这样将 208 赋给 DT：

```
DT.X=208
```

而不是用 X=208 或 DTs(X)值域=X 的形式。

点“.”告诉程序有一个叫做 X 的数据域存在。上面的例子中还定义了一个叫做 Y 的数据域，因此也可以给它赋一个值：

```
DT.Y=387
```

用户定义类型(Type)必须放在一个窗体(或 .BAS 文件)模块的 Declaration(声明)段中。

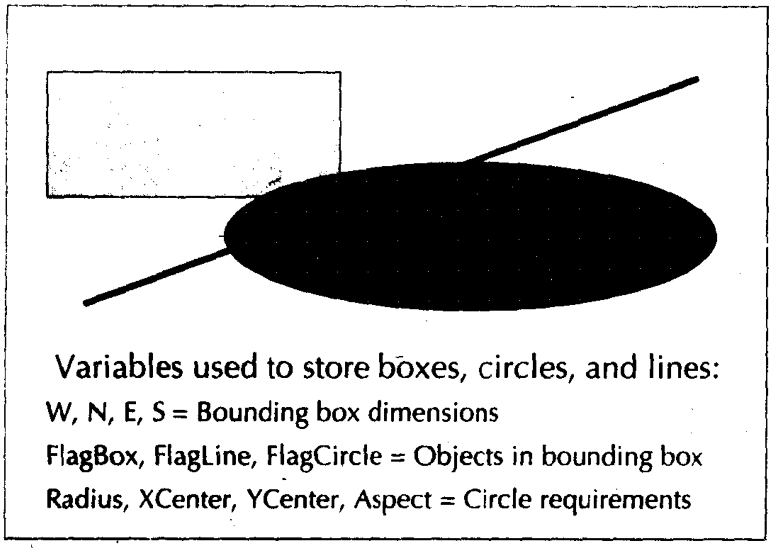
2.15.3　为 Artisan 程序声明变量

清单 2.5 显示出 Artisan 程序的 Declaration(声明)段划分成三个部分。程序中用到的其他全局变量与常量还需要参阅 ARTISAN7.BAS 文件(附录 A)的模块声明及过程。

清单 2.5　Artisan 主窗体的声明段

```
Declarations
DefInt A-Z
Const Limit = 1000
Dim Draw As Integer
Dim FillValue(Limit) As Integer 'No Fill
Dim LineValue(Limit) As Integer 'No Line
Dim OFFSETx, OFFSETy, PtInRectBasic As Integer
Dim WIDE, HIGH As Integer 'Monitor Type
Dim PixelX, PixelY As Integer
Dim SSM As Single 'Side Scroll Move
Dim BSM As Single 'Bottom Scroll Move
Dim LastMove As Integer
Dim LastMenu As Integer
Dim FlagMove As Integer
Dim V As Integer 'Ruler Guide Lines
Dim TGL As Integer
Dim NewGuideLine As Integer
Dim GuideX(Limit) As Long
Dim GuideY(Limit) As Long
Dim Zoom As Integer 'Zoom Offsets
Dim NodeZoom As Integer
Dim Scaling As Integer
Dim OldSW As Double ' Scales
Dim OldSH As Double
Dim OldRH As Single ' Rulers
Dim OldRW As Single
Dim PerSW As Single ' Percentage of scales
Dim PerSH As Single
Dim Xstr As String, Ystr As String
Dim OutLineColor(Limit) As Long
Dim PaintColor(Limit) As Long
Dim LineWidth(Limit) As Integer
Dim TextX, TextY As Integer ' Text Values
Dim FlagAlign(Limit) As Integer
Dim OFFSET(Limit) As Integer
Dim TextSize(Limit) As Single
Dim TextFont(Limit) As String
Dim TextBold(Limit) As Integer
Dim TextItalic(Limit) As Integer
Dim Storedtext(Limit) As String
```

图 2.3 给出了一些标准对象属性。



Variables used to store boxes, circles, and lines:

W, N, E, S = Bounding box dimensions

FlagBox, FlagLine, FlagCircle = Objects in bounding box

Radius, XCenter, YCenter, Aspect = Circle requirements

图 2.3 一些标准对象属性

DefInt A－Z：指定所有名字为 A 至 Z 的变量默认为整数。

Const Limit：本程序中索引表仅允许画 1000 个对象或边界方框。可以将此值加大，但是，会受到 Visual Basic 数据限制或内存的限制。

Draw：True 或 False，用来在每次快速重画时跳过 UpdateObjects。

FillValue：每个对象的 FillStyle 值，通常是 0（实心填充）或 1（透明）。

LineValue：对象的 DrawStyle 设置值。因为 DrawWidth 不能等于 0，因此，一个没有边框的对象必须设置为 DrawStyle 5（不可见）才能获得所要的效果。

OFFSETx，OFFSETy，PtInRectBasic：与 MoveObject 过程一起使用。

WIDE 和 HIGH：用来切换监视器分辨率。VGA 和 SVGA 通常为每英寸 96 个象素。

PixelX 和 PixelY：代表 Visual Basic 语言参考指南中定义的 TwipsPerPixelX 和 TwipsPerPixelY。

SSM(侧边滚动移动)和 BSM(底边滚动移动)：当移动 ArtWork 及 Rulers 图片框时滚动条所用的位移值。

LastMove：当 UpdateObjects 循环重画所有对象时，区分最后哪一个对象的图柄是可见的。

LastMenu：单击鼠标右键时显示哪个菜单栏。当单击鼠标右键时，最后选定的菜单就会显示出来。

FlagMove：告知程序你正在移动一个对象，因此不能画新的对象。它的值为 True/False。

V(值)，TGL(总标线)，NewGuideLine，GuideX 及 GuideY：用于存储下拉式标尺线的 X，Y 位置。

Zoom-PerSH：在进行了一次缩放之后用来重新确定控制对象比例的人工位移。

Xstr 和 Ystr：用来显示鼠标在屏幕上移动时 X，Y 坐标的串变量。

OutLineColor：用来绘制对象边框的颜色索引值。

PaintColor：用于填充对象内部颜色的索引值。

LineWidth：对象边框的宽度。所有对象的 LineWidth 至少必须是 1 个单位宽，如果一个对象没有边框，就必须用 LineValue 将其标记为透明的。

TextX 和 TextY：用于绘制第一行文字的当前 X，Y 位置。因为 CurrentX 和 CurrentY 具有同样的功能，所以这些值并不是必需的。TextX 和 TextY 只是用来增加一些程序的保险系数。

FlagAlign：用于按左、中、右方式将文字对齐。值分别为 0＝左对齐，6＝居中对齐，2＝右对齐。

OFFSET：使用 TextWidth 值，这样，FlagAlign 就能正确地定位到文字光标的 CurrentX 和 CurrentY 处。

TextFont：方框中键入字符串的字体名。

TextSize：键入字符串中每个字符的尺寸。

TextBold：标记文字的粗细，可以是 on(开)或 off(关)。

TextItalic：标记文字是否斜体，可以是 on(开)或 off(关)。

StoredText：存储每一个文字对象的全部文字串。

图 2.4 给出了一些标准文字对象的属性。

```
Dim B As Integer ' Current Object (Bounding Box)
Dim Total As Integer ' Total Number of Objects
Dim X1 As Long
Dim Y1 As Long
Dim X2 As Long
Dim Y2 As Long
Dim XX1 As Long
Dim YY1 As Long
Dim XX2 As Long
Dim YY2 As Long
Dim W(Limit) As Long 'West or Left
Dim E(Limit) As Long 'East or Right
Dim N(Limit) As Long 'North or Top
Dim S(Limit) As Long 'South or Bottom
Dim FlagBézier(Limit) As Integer
Dim FlagLine(Limit) As Integer
Dim FlagBox(Limit) As Integer
Dim FlagCircle(Limit) As Integer
Dim FlagText(Limit) As Integer
Dim Radius As Integer
Dim XCenter As Single
Dim YCenter As Single
Dim Aspect As Single
Dim SideRuler As Integer
Dim TopRuler As Integer
Dim X1Ruler As Integer
Dim Y1Ruler As Integer
Dim X2Ruler As Integer
Dim Y2Ruler As Integer
Dim DeleteObject As Integer 'DELETE
'Bézier Curve
Declare Function Polyline Lib "GDI" (ByVal hDC%, P As POINTS,
ByVal NUM_PTS%) As Integer
Const MAX_POINTS = 33
Dim P_Index(1 To MAX_POINTS) As POINTS
Dim BezErase As Integer
Dim ShowDotted As Integer
Dim p As Integer
```

Attributes of this text if it

had been typed on the screen

FlagAlign = 6(centered); OFFSET=TextWidth\2

TextFont=Lydian;TextSize=18 point

TextBold=True;TextItalic=False

StoredText=显示的字符串

W,N,E,S=边框的位置

PaintColor=字体颜色

FlagText=边框中对象的类型

图 2.4 标准文字对象属性

Dim B:这是本程序中主要的也是最重要的变量,因为它代表一个对象第一次通过绘制生成时所给定的编号。每个对象位于某个边界方框中,因此我将此变量取名为 B(代表方框)。

Dim Total:已绘制的边界方框总数。通过不断增加 Total 与 B 的值即可获取到目前为止所画边界方框对象的总数。

B=Total+1:每当绘制一个新对象时,该语句将编号增 1。下一条语句,Total=B,确认

了这一点。例如，假设绘制的第 3 个对象恰好是一个矩形，那么所有与该对象有关的索引标题都应该等于 B，所有的值都应该存在索引号 3 中。与 3 号矩形相关的属性中包括 OutLienColor(3)，paintColor(3)以及它在屏幕上的位置(W(3)，N(3)，E(3)和 S(3))。

X1，X2，Y1，Y2，XX1，XX2，YY1，YY2：当一个对象移至新位置，或绘制一个新对象时，用来擦除上一个对象位置的临时屏幕坐标。

W，N，E 和 S：本程序中第二重要的变量。它们代表每一个对象所在边界方框的四个位置。W 代表方框的西边，N 代表北边，E 代表东边，S 代表南边。每一个值代表了方框的对角线角点，但这在视觉上可能让人有些混淆。这些值也可以互换，这取决于起始(例如，MouseDown 开始画图)和终止(或 MouseUp)的方向，或者是否翻转一个对象。

FlagBézier，FlagLine，FlagBox，FlagCircle 和 FlagText：值为 TRUE 或 FALSE 的标记，用来更新对象。例如，如果所画的第四个对象是一个圆，FlagCircle 索引应被设成 4。请记住，一旦画完了第四个对象后，B 就等于 4 了。UpdateObjects 过程(清单 2.6)根据这些变量名就可以知道必须重画哪些对象。

清单 2.6　UpdateObjects 过程中所用标志的名字

```
Sub UpdateObjects ()
LastMove = B
ArtWork.DrawMode = Copy_Pen
ArtWork.DrawStyle = Solid
ArtWork.AutoRedraw = True
If Draw = True Then GoTo SkipErase
ArtWork.DrawWidth = 20
ArtWork.Line (W(B), N(B))-(E(B), S(B)), White, BF
ArtWork.Line (XX2, YY2)-(XX1, YY1), White, BF: B = 1
ArtWork.DrawWidth = 1
    UpdateDeskTop
Do While B < Total + 1
  ArtWork.DrawWidth = LineWidth(B)
  If LineValue(B) > 0 Then ArtWork.DrawStyle = 5 Else ArtWork.DrawStyle = Solid
SkipErase:
Select Case B
  Case FlagCircle(B)
    X1 = W(B): Y1 = N(B): X2 = E(B): Y2 = S(B)
    GoodCircle = PerfectCircle(X1, Y1, X2, Y2)
    ArtWork.FillColor = PaintColor(B): ArtWork.FillStyle = FillValue(B)
    ArtWork.Circle (XCenter, YCenter), Radius, OutLineColor(B), , , Aspect
    ArtWork.FillColor = 0: ArtWork.FillStyle = 1
  Case FlagLine(B)
    ArtWork.Line (W(B), N(B))-(E(B), S(B)), OutLineColor(B)
  Case FlagBox(B)
    If FillValue(B) = 1 Then GoTo SkipFill
    ArtWork.Line (W(B), N(B))-(E(B), S(B)), PaintColor(B), BF
 SkipFill:
    ArtWork.Line (W(B), N(B))-(E(B), S(B)), OutLineColor(B), B
  Case FlagBezier(B)
    p1.X = P1X(B): p1.Y = P1Y(B): p2.X = P2X(B): p2.Y = p2Y(B)
    h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
    ArtWork.ForeColor = OutLineColor(B)
    BezDraw
   Case FlagText(B)
    ArtWork.CurrentX = OFFSET(B): ArtWork.CurrentY = N(B)
    SaveScaleLeft = ArtWork.ScaleLeft
    ArtWork.ScaleLeft = -ArtWork.CurrentX + Scaling
    ArtWork.FontName = TextFont(B)
    ArtWork.ForeColor = PaintColor(B)
    ArtWork.FontSize = TextSize(B) * TextPoint
    ArtWork.FontBold = TextBold(B)
    ArtWork.FontItalic = TextItalic(B)
    wFlag = FlagAlign(B)
    TA = SetTextAlign(ArtWork.hDC, wFlag)
    ArtWork.Print Storedtext(B)
    ArtWork.ScaleLeft = SaveScaleLeft
End Select
'''''
If Draw = True Then Draw = False:
    ArtWork.AutoRedraw = False: B = LastMove: Exit Sub
  B = B + 1
Loop
B = LastMove
  ArtWork.AutoRedraw = False
  ArtWork.DrawWidth = 1
  ArtWork.DrawStyle = Solid
If DeleteObject = True Then Exit Sub
If CurrentTool = 1 Or CurrentTool = 2 Then Exit Sub
If SideRuler = True Or TopRuler = True Then Exit Sub
  Handles
  Nodes
End Sub
```

Radius，XCenter，YCenter 和 Aspect：用 Circle 语句生成圆或椭圆时所需的值。例如：circle(XCenter，YCenter)，Radius，，，Aspect。

SideRuler，TopRuler：是否拉动了两个标尺中的标线，如果拉动了，该值设成 True，禁止在屏幕上画图(MouseMove 事件可能引起画图操作)。也可以使用 ArtWork. Enabled=False。

X1Ruler，Y1Ruler，X2Ruler，Y2Ruler：用于画标尺标线的临时线位置。用作 Line 语句中的 X1，Y1，X2，Y2。

DeleteObject：用来退出 UpdateObjects，这样在一个对象被删除了以后，该对象的图柄和节点都不会显示出来。

Declare Function Polyline：一个画直线的 Windows API 函数，该函数与使用 Step 关键字的 Line 语句或 DOS 版本 Basic 语言中的 DRAW 语句类似。

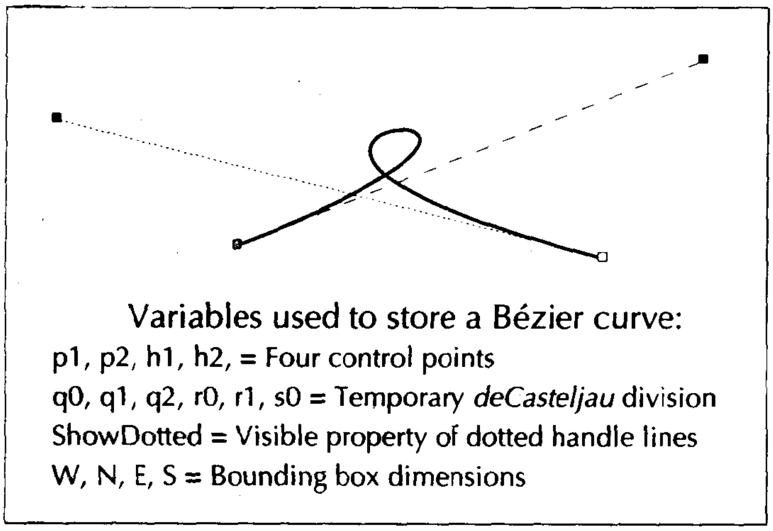
Const MAX_POINTS：一条 Bézier 曲线上存储的总点数。

P_Index：保存 Bézier 点的索引。每条曲线的控制点都是用一个叫做 POINTS 的 Type 变量存储，该变量将 X 和 Y 坐标组合起来，使之只占用一个索引号。

BezErase：这是一个标记，该标记用来擦除移动中的 Bézier 曲线的第一幅图象。

Showdotted：决定是否显示 Bézier 图柄线(点线)，真(True)表示显示点线。

P：索引数组的起始点。



Variables used to store a Bézier curve:
p1, p2, h1, h2, = Four control points
q0, q1, q2, r0, r1, s0 = Temporary *deCasteljau* division
ShowDotted = Visible property of dotted handle lines
W, N, E, S = Bounding box dimensions

图 2.5　绘制 Bézier 曲线所需的标准属性

图 2.5 显示了绘制一条 Bézier 曲线所需的标准属性。

```
Dim p1 As POINTS    ' Bézier Control Points / Handles
Dim h1 As POINTS
Dim h2 As POINTS
Dim p2 As POINTS
Dim p1X(Limit) As Long
Dim p1Y(Limit) As Long
Dim p2X(Limit) As Long
Dim p2Y(Limit) As Long
Dim h1X(Limit) As Long
Dim h1Y(Limit) As Long
Dim h2X(Limit) As Long
Dim h2Y(Limit) As Long
Dim C1(1 To MAX_POINTS) As POINTS
Dim D1(1 To MAX_POINTS) As POINTS    'Curve Points
Dim D2(1 To MAX_POINTS) As POINTS
Dim D3(1 To MAX_POINTS) As POINTS
Dim E1(1 To MAX_POINTS) As POINTS
Dim E2(1 To MAX_POINTS) As POINTS
Const YELLOW = &HFFFF&
```

p1，h1，p2，h2：Bézier 曲线的 4 个控制点。P1＝Bézier 节点 1，P2＝Bézier 节点 2，H2＝Bézier 图柄 1，H2＝Bézier 图柄 2。它们以一种叫做 Points 的数据类型存储，该数据类型保存 X，Y 坐标点。若想取得每一个值，请使用结构 P1.X，P1.Y，P2.X，P2.Y，H1.X，H1.Y，H2.X，H2.Y。如果线中所用的节点 1(P1)位于 X 坐标 1004，Y 坐标 2205 处，则 P1.X 等于 1004，P1.Y 等于 2205。不能将这些值作为参数传给 Visual Basic 的过程(如 Sub BezDivide P1.X%，P1.Y%)。不过可以写成 Dim P1 As Points，然后将参数定义成下面的样子：

```
Sub BezDivide(P1 As Points,...)
```

p1x,p1y,p2x,p2y,h1x,h1y,h2x,h2y:这些值用来存储位于 POINTS 类型结构中的 X,Y 位置,要用这些值更新递归的 Bézier 曲线点。

D1,D2,D3,E1,E2,C1:Bézier 曲线的实际划分点。最有意义的是 C1,它是每次笛卡尔划分后实际的曲线点。当算出了 33 个这样的点后,曲线就画出来了(参见图 2.6)。用 Line 语句或 API 函数 Polyline 将这些点连起来。更多的信息,请参见第 3 章中的 Bézier 曲线一节。

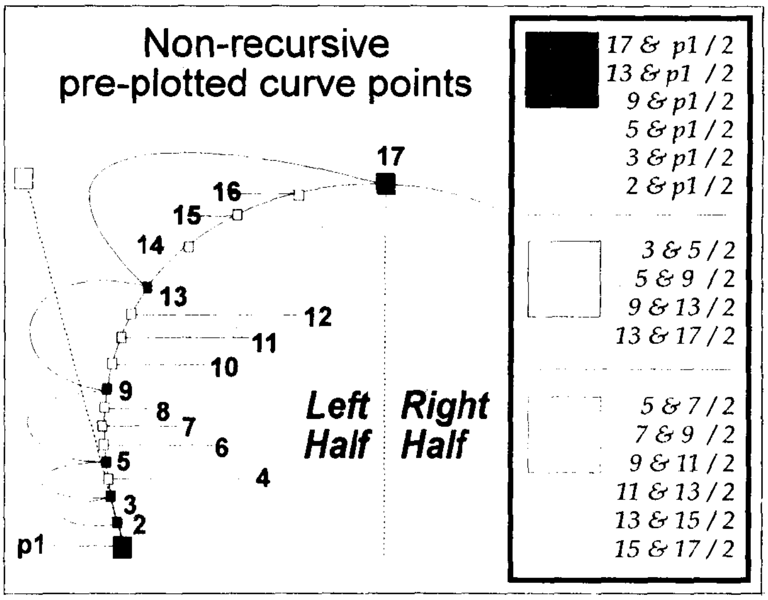

Non-recursive
pre-plotted curve points
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
p1
Left Half
Right Half
17 & p1 /2
13 & p1 /2
9 & p1 /2
5 & p1 /2
3 & p1 /2
2 & p1 /2
3 & 5 /2
5 & 9 /2
9 & 13 /2
13 & 17 /2
5 & 7 /2
7 & 9 /2
9 & 11 /2
11 & 13 /2
13 & 15 /2
15 & 17 /2

图 2.6 绘制曲线点的顺序

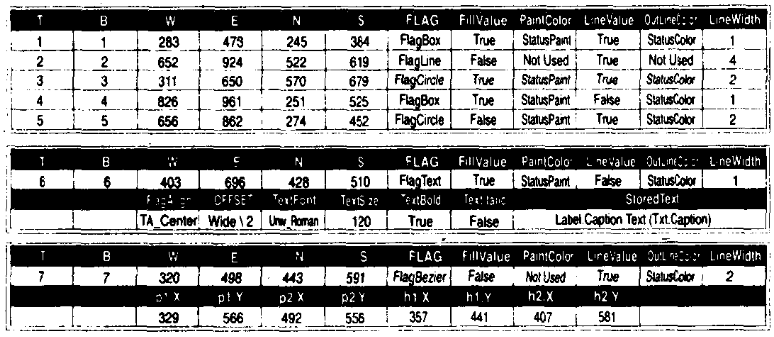
Const Yellow:表示黄颜色的十六进制编号。移动一个曲线图柄时所看到的点线是青色的,当将 DrawMode 设置成以 Xor(异或)模式画象素时,黄色看上去就像是青色。Xor 模式可以恢复点线移动时的背景颜色。

附注: 每一个图表标题代表一个变量名。变量 B 的值指定索引号(对象编号),并用属于该对象的正确属性值填充此数组单元。你可以定义一个巨型的多维数组来处理所有的东西,或定义一个大的自定义 Type 结构。使用 API 语句时也可以删去本程序中所用的某些数组。

2.15.4 启动时默认的窗体装载值

为了能在第一次装入窗体或窗体大小发生改变时将所有控制对象放置在正确的位置,应该确定出所用监视器的类型。我使用的是一台 15 英寸 800×600 分辨率的 SVGA 监视器(不到 12000 个 twip 宽)和一台 19 英寸 1664×1200 分辨率的 Hercules(大力神)监视器,该监视器超过 19000 个 twip 宽。

一个 twip 是指每英寸 1440 个单位,大多数 VGA 类型的监视器都是每英寸 96 个象素(1440\15,或 96),而有些双页监视器是每英寸 118 个象素(大约 1440\12,或 120)。可以用



T	B	W	E	N	S	FLAG	FillValue	PaintColor	LineValue	OutLineColor	LineWidth
1	1	283	473	245	384	FlagBox	True	StatusPaint	True	StatusColor	1
2	2	652	924	522	619	FlagLine	False	Not Used	True	Not Used	4
3	3	311	650	570	679	FlagCircle	True	StatusPaint	True	StatusColor	2
4	4	826	961	251	525	FlagBox	True	StatusPaint	False	StatusColor	1
5	5	656	862	274	452	FlagCircle	False	StatusPaint	True	StatusColor	2

T	B	W	E	N	S	FLAG	FillValue	PaintColor	LineValue	OutLineColor	LineWidth
6	6	403	696	428	510	FlagText	True	StatusPaint	False	StatusColor	1
		FlagAlign	OFFSET	TextFont	TextSize	TextBold	TextItalic	StoredText			
		TA_Center	Wide \ 2	Unv_Roman	120	True	False	Label.Caption Text (Txt.Caption)			

T	B	W	E	N	S	FLAG	FillValue	PaintColor	LineValue	OutLineColor	LineWidth
7	7	320	498	443	591	FlagBezier	False	Not Used	True	StatusColor	2
		p1 X	p1 Y	p2 X	p2 Y	h1 X	h1 Y	h2 X	h2 Y		
		329	566	492	556	357	441	407	581		

图 2.7　7 个对象的属性数组

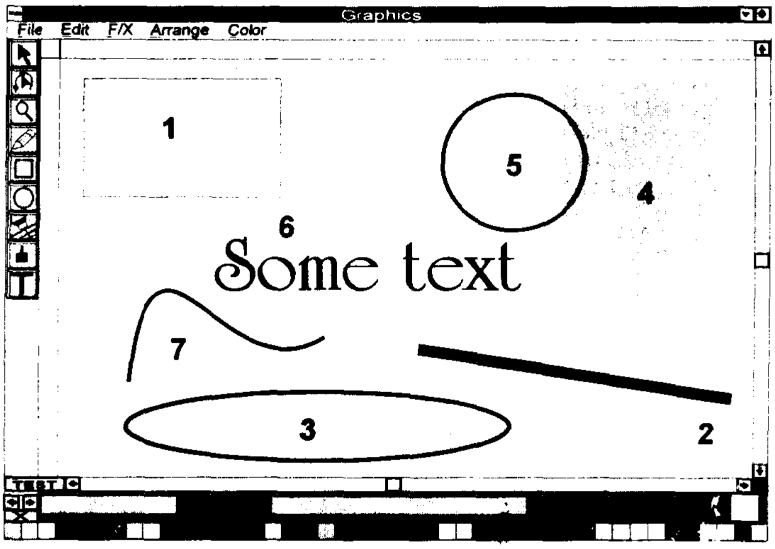


图 2.8　使用图 2.6 中的值时 7 个对象的外观

19000 个单位作为大多数 VGA 监视器的 ScreenWidth 的基值，也可以借助于在 API GetDeviceCaps 函数调用中通过传递 LogPixelsX 和 LogPixelsY 参量求出每英寸逻辑象素数。

下面是一个使用 API GetDeviceCaps 的例子：

```
Declare Function GetDeviceCaps% Lib "GDI"(ByVal hDC%, ByVal nIndex%)
Const LOGPIXELSX=88:Const LOGPIXELSY=90
```

按下面的示例语句调用 API GetDeviceCaps 函数：

```
X=GetDeviceCaps(Form1.hDC,LOGPIXELSX)
Y=GetDeviceCaps(Form1.hDC,LOGPIXELSY)
Print X: Print Y
```

清单 2.7　程序开始时的初始值

```
Sub Form_Load ()
PixelX = Screen.TwipsPerPixelX: PixelY = Screen.TwipsPerPixelY
WIDE = Screen.Width \ PixelX: HIGH = Screen.Height \ PixelY
```

```
    NewGuideLine = False
    ArtWork.FontSize = 12: txt.FontSize = 12
    ArtWork.FontName = txt.FontName
    WindowState = 2
    TextPoint = 1
    PerSH = 1: PerSW = 1
    StatusColor.BackColor = Black
    LineWidth(B) = 1
    PenColor = Black
    MoveFlag = False
    LastMenu = 3
    Convert_Curve = False
    Convert_Line = False
    P = 1: B = 0: Total = 0
    ArtWork.Enabled = False
    ArtWork.BackColor = White
    ArtWork.FillColor = White
    ArtWork.FillStyle = TransParent
    VScroll.Value = 0: HScroll.Value = 0
    Color_INI
    Load TextForm
    KeyPreview = True
End Sub
```

为了绘图的目的，最好是使用 twip 作为 ScaleMode 而不要用象素或者其他度量单位。不过大多数 API 过程都要求以象素作为度量单位。你可以将 twip 转换为象素，方法是将 twip 单位除以 15 或 12，这取决于监视器的类型，或者用 TwipsPerPixelX 和 TwipsPerPixelY 方法。

PIXELX 和 PIXELY：TwipsPerPixelX 和 TwipsPerPixelY 语句可以完成这种转换，并可以和 Screen.Width 及 Screen.Height 一起用来获取大多数监视器的每英寸象素值。如果不能，请参见 API GetDeviceCaps 函数。

NewGuideLine：从标尺拉出的一条点标线。该行用来计算更新对象时屏幕上的标线数。

ArtWork.FontSize：将桌面默认字体大小设为 12 点。

Txt.FontSize：将 TxtLabel 的默认字体大小设为 12 点。

WindowState：将主窗体最大化，使其充满整个屏幕。

TextPoint：当用 ZoomTool（缩放工具）来放大（Zoom In）或缩小（Zoom Out）时，该值被用来乘以文字的点大小。在放大或缩小时，TextPoint 值是一个大约的放大或缩小的百分比。实际所用值的精度是十万分之一。用值 1 乘以该偏移不会引起任何变化，如清单 2.8 所示。

清单 2.8　ZoomBar 过程的部分清单

```
Sub ZoomBar_MouseDown (Button%, Shift%, X!, Y!)
Select Case CurrentZoom
  Case ZoomIn 'First Button
    If Zoom_Out = True Or Zoom_In = True Then Beep: Exit Sub
    ArtWork.AutoRedraw = True: ArtWork.Cls: ArtWork.AutoRedraw = False
    Scaling = 3750: Zoom_In = True: Zoom_Out = False
 * * * Approximate enlargement of text point size * * *
    Zoom = -48: NodeZoom = -21: TextPoint = 1.7
    txt.FontSize = txt.FontSize * TextPoint
    ZoomScale 2, 3750, 3750, -7500, -6375
  Case ZoomOut 'Second Button
    If Zoom_Out = True Or Zoom_In = True Then Beep: Exit Sub
    ArtWork.AutoRedraw = True: ArtWork.Cls: ArtWork.AutoRedraw = False
    Scaling = -5000: Zoom_In = False: Zoom_Out = True
 * * * Approximate reduction of text point size * * *
    Zoom = 65: NodeZoom = 28: TextPoint = .65
    txt.FontSize = txt.FontSize * TextPoint
    ZoomScale 2, -5000, -5000, 10000, 8500
  Case ZoomAll 'Third Button
  Case ZoomPage 'Fourth Button
    ArtWork.AutoRedraw = True: ArtWork.Cls: ArtWork.AutoRedraw = False
    ArtWork.Scale: PerSW = 1: PerSH = 1
 * * * reset the TextPoint value to 1,normal text point size * * *
    Zoom = 0: NodeZoom = 0: TextPoint = 1
    Scaling = 0: Zoom_In = False: Zoom_Out = False
'Note:Partial listing only!
  ZoomScale 2, 0, 0, 0, 0
End Select
End Sub
```

PerSH 和 PerSW：用来除比例(scale)单位。默认值 0 会引起“被 0 除”错误。

StatusColor.BackColor：将画笔的颜色初始化为默认的黑色。

LineWidth(B)：启动时，该值设定为 1。LineWidth(B)等于 0 或较小会产生错误，原因是该变量也用来设置画线宽度。DrawWidth 属性值不能小于 1。

PenColor(0)：将画笔索引号初始化为 0(否则会产生运行期间错误)。

MoveFlag：目前没有对象处于移动中，因此该值设为 False。

LastMenu：单击鼠标右键时显示 Arrange 菜单。

ConvertCurve：没有直线正在转换成曲线，因此该标志被关掉。

P=1:B=0:Total=0：对象个数初始化为 0(即没有对象)。

ArtWork.Enabled：鼠标不会与 Rulers_Paint 事件冲突。

ArtWork.BackColor：绘图区被设成白色。

ArtWork.FillColor 和 ArtWork.FillStyle：第一个所画对象的默认值。例如，方框被填充成白色，然后设成透明或空心的。

VScroll.Value 和 HScroll.Value：滚动框被置于每个滚动条的中间。

Color_INI：在 ColorBar 调色板中装入默认颜色。

Load TextForm：将 TextForm 装入内存中，因为大量的字体会降低程序运行的速度。

KeyPreview：键盘事件首先被窗体调用，然后再被聚焦的控制对象调用(例如，KeyDown，KeyUp，KeyPress)。

2.16　确定控制对象在 Artisan 窗体中的位置

清单 2.9 中给出的 Form_Resize 事件用来在程序启动时确定要显示的对象位置和大

小。待定位的控制对象从其名字就可以一目了然(例如,Ruler(0),Ruler(1)和 ToolButtons)。PictureMask 图片框控制对象用作 ArtWork 图片框的载体。这样就可以滚动 ArtWork 窗口中的任意图片。被屏蔽的控制对象即使当窗体在显示时仍是不可见的。

Move 语句用来重定位各项控制对象,它的结构如下:

```
object. Move Left, Top, Width, Height
```

清单 2.9 Form_Resize 重定位 Artisan 窗体上的所有控制对象

```
Sub Form_Resize ()
ToolButtons. ZOrder : StatusArea. ZOrder : ColorMask. ZOrder
'StatusArea picture box''''''
StatusArea. Visible = False
StatusArea. Move 0, 0, WIDE, 70
ColorMask. Move 0, 50, WIDE, 20
ColorBar. Move 0, 0, WIDE, 20
StatusColor. Move WIDE - 36, 20, 28, 28
StatusPaint. Move 2, 2, 24, 24
XYlabel. Move 36, 20, WIDE \ 5, 26
StatusLabel. Move (WIDE \ 2) - (StatusLabel. Width \ 2), 20, WIDE \ 3, 26
ColorScroll. Move 0, 20, 34, 17
No_Fill. Move 0, 37, 34, 13
TestButton. Move 0, 0, 56, 17
StatusArea. Visible = True
''''''''''''''''''''''''''''''

ZoomBar. Move 64, ToolButtons. Top + 70, 128, 34
LineButtons. Move 64, ToolButtons. Top + 192, 256, 34
PaintButtons. Move 64, ToolButtons. Top + 224, 256, 34
PictureMask. Move -200, -250, WIDE + 400, HIGH + 400
ArtWork. BorderStyle = 0
ArtWork. Move 0, 0, PictureMask. Width, PictureMask. Height
RulerCross. Move 40, 0, 22, 22
Ruler(0). Move 40, PictureMask. Top, 22, PictureMask. Height 'Side Ruler
Ruler(1). Move PictureMask. Left, 0, PictureMask. Width, 22'Top ruler
Txt. Move 0, 0
ToolButtons. Move 0, 0, 40, HIGH
Vscroll. Move WIDE - 17, 22, 17, StatusArea. Top - 22
Hscroll. Move 56, 0, WIDE - 73, 17

ScaleBox. Move 3600, 3750, 330, 330
OldSW = ArtWork. ScaleWidth: OldSH = ArtWork. ScaleHeight
OldRW = 300: OldRH = 300

Ruler(0). Scale: Ruler(1). Scale
UpdateObjects
CurrentTool = 0
End Sub
```

ToolButton. ZOrder:ZOrder 语句使得 ToolButtons、StatusArea 和 ColorMask 控制对象都显示在桌面上,它们不会被标尺或桌面绘图区遮挡。

当改变控制对象的大小时,它们的 Visible(可见)属性都被设成 False,所以看不到在定位这些控制对象时的混乱局面。在 Resize 或 Paint 事件中千万不要将控制对象设成可见的,

因为这样做会使这些事件被调用两次。

图 2.9 显示的是运行完 Form_Resize 后屏幕的样子。

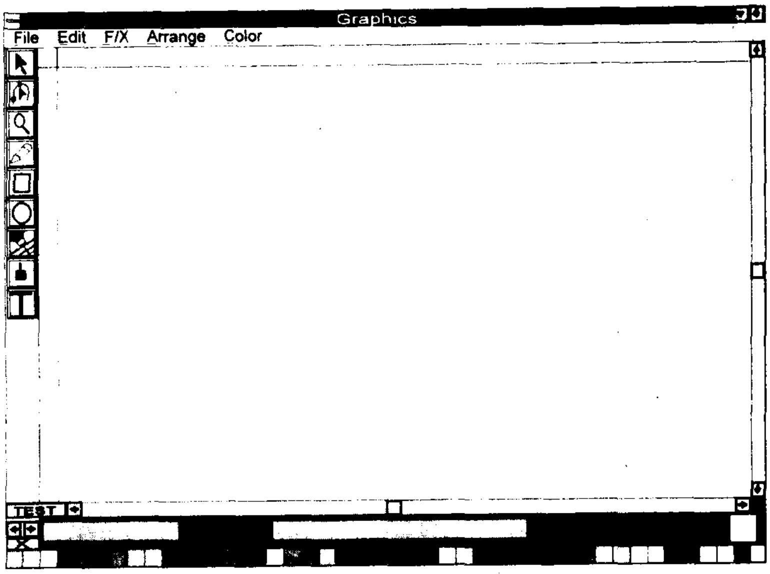


图 2.9　运行完 Form_Resize 后的屏幕显示

2.17　从工具箱选定一个绘图工具

你可以从 ToolButtons 控制中选定一种绘图工具。单击某个按钮后，该按钮的图象被反显，给人一种按钮被按下去的感觉。要实现这种效果有许多方法，附带盘中 16MISC 目录下的 Button 程序给出了 3 个例子。

ToolButton_MouseDown 的其余部分用来隐藏那些在选定了不同的工具时可见的控制对象。例子请看清单 2.10。

清单 2.10　ToolButton_MouseDown 改变按钮的颜色，使其看上去像是被按下了

```
Sub ToolButtons_MouseDown (Button%, Shift%, X As Single, Y As Single)
ArtWork.ForeColor = Black
ArtWork.MousePointer = 0
LastToolUsed = CurrentTool: NewTool = Y \ 32
  If Y < 288 Then
    ToolButtons.Line (0,CurrentTool * 32+1)-(31,CurrentTool * 32+31),,BF
    ToolButtons.Line (0, NewTool * 32 + 1)-(31, NewTool * 32 + 31), , BF
    CurrentTool = NewTool
End If
Select Case CurrentTool
  Case 1
   If BezHandle1.Visible Then Exit Sub
   If FlagBezier(B) = B Then ShowDotted = True: BezUpdate
  Case 2
   ZoomBar.Visible = True
  Case 6
```

```
    LineButtons. Visible = True
   Case 7
    PaintButtons. Visible = True
   Case 8
    ArtWork. MousePointer = 2 'Cross
 End Select
 If LastToolUsed = TextTool Then ReDrawText
 If CurrentTool = 0 And Total > 0 Then
   Nodes
   Handles
 Else
   For R = 0 To 7
     Handle(R). Visible = False
   Next R
 End If
 If CurrentTool <> 1 And BezHandle1. Visible Then ShowDotted = [CI:CCC]
     False: BezUpdate
   If CurrentTool <> 2 Then ZoomBar. Visible = False
   If CurrentTool <> 6 Then LineButtons. Visible = False
   If CurrentTool <> 7 Then PaintButtons. Visible = False
 End Sub
```

ArtWork. ForeColor：ForeColor 是当前画笔的绘图颜色，在画另一个对象之间总要被重设成黑色，否则新对象的 OutlineColor（边框颜色）就会是上一次用 ColorBar 调色板选定的颜色。

LastToolUsed：当选定了一种工具后，该标志用来更新 ZoomBar，LineButtons 和 PaintButtons 失去焦点时的 Paint 事件。

NewTool：Y 是鼠标在 ToolButtons 上面移动时的当前 Y 位置。每个绘图图标是 32 个象素高，图标总数(9)乘以它们的高度(32)就是 288。用公式 Y/32 就可以返回每个按钮图标的准确按钮号(0－9)。

CurrentTool＝NewTool：记录当前工具，执行与该工具相关的代码。例如，如果按下了方框图标按钮，鼠标 Y 值应该在 128 至 160 之间。第一条 Line 语句将上一个工具反转过来，使其恢复原来的样子，下一条 Line 语句反转该按钮图标图片。最后，CurrentTool＝NewTool 告知下面的 Select/Case CurrentTool 语句应该执行哪一段代码。

Case 1：BézierTool(Bézier 工具)被选中。如果 Bézier 图柄是可见的，连接该图柄和 Bézier 节点的点线也应该是可见的。这时不进行任何操作。如果图柄是不可见的，点线就不连接任何东西。如果 showdotted＝True，则调用 BezUpdate 擦去此线。

Case 2：ZoomBar 当前是不可见的，因此应将其显示出来。

Case 6：LineButtons 当前是不可见的，因此应将它们显示出来。

Case 7：PaintButtons 当前是不可见的，因此应将它们显示出来。

Case 8：TextTool 被选定，所以将光标改为十字形状。

下面的部分处理 ToolButton_MouseDown 事件，该事件使 ArtWork 失去聚焦（如果你单击了 ToolButton）。例如，如果选定了 TextTool（文字工具）而且用户在窗口中输入了某些内容，那么，在选定另一个工具时就应该更新这些文字，任何可见的图柄或节点也都应该隐藏起来。

If LastToolUsed = TextTool：这意味着你很有可能输入了一些文字，因此应该用 RedrawText 过程设定所有输入内容的属性。

If CurrentTool = 0 And Total>0 Then：选定了指针工具，所以应该高亮显示当前对象，显示出该对象周围的节点和图柄，或者，如果选定的不是指针工具，就没有必要显示出对象周围的图柄。

If CurrentTool<>(1)：检查是否要擦除 Bézier 曲线的点线。

If CurrentTool<>(2,6,7)：选定了 ZoomBar，LineButtons 及 PaintButtons 中的一种后，应该将其他的隐藏起来。

2.18 在窗体第一次启动时就按下某个按钮

清单 2.11 中给出了 ToolButtons_Paint 事件在所选定的工具按钮上画一个反显的方框，给人一种该按钮被按下去了的感觉。

清单 2.11 重画一个反转了的工具按钮，恢复其原来的形状

```
Sub ToolButtons_Paint ()
    ToolButtons.Line (0, CurrentTool * 32 + 1)-(31, CurrentTool * 32 + 31), , BF
End Sub
```

CurrentTool 表示从 ToolButton 上选定的画图工具。每当选定一个新的工具时，CurrentTool 就等于指定给该按钮名的索引号。按钮名及其相应的值列在 ARTISAN7.BAS 文件声明段的标题 Artisan's ToolPalette IDs 之下。值的范围从 0(顶部按钮)到 8(底部按钮)。

2.19 单击窗口中的工具按钮

单击 ArtWork 图片框时，节点和图柄显示与否取决于是否触到了某个对象。程序会跳转到与当前选定工具相关的过程中(参见清单 2.12)。

清单 2.12 ArtWork_MouseDown 启动桌面上的所有绘图操作

```
Sub ArtWork_MouseDown (Button%, Shift%, X As Single, Y As Single)
ZoomBar.Visible = False
LineButtons.Visible = False
PaintButtons.Visible = False
If SideRuler = True Or TopRuler = True Then Exit Sub
If Button = Left_Button Then N_H_Clear
Select Case CurrentTool
  Case PointerTool
    Find_Object X, Y
    BoundingBox_Down X, Y
  Case BézierTool
    If BezHandle1.Visible = True Then ShowDotted = False: BezUpdate
    BoundingBox_Down X, Y
  Case LineTool To CircleTool
    BoundingBox_Down X, Y
  Case TextTool
```

```
        If txt.Caption > "" Then Align = 0: ReDrawText
        If Button = Left_Button Then

           txt.BorderStyle = 0
           Dim tm As TEXTMETRIC
           API = GetTextMetrics(ArtWork.hDC, tm)
           Leading = (tm.tmExternalLeading + tm.tmInternalLeading) * PixelY
           txt.Left = X: txt.Top = Y - (txt.Height - Leading)
        End If
    End Select
    If CurrentTool <> 1 Then BezNode1.Visible = False: BezNode2.Visible = False
    End Sub
```

如果因为通过 ToolButtons 选定了 ZoomBar，LineButtons 或者 PaintButtons 而使它们可见，但又未在这些弹出工具上作任何选定，那么必须重新将它们隐藏起来。也可以在 Lost Focus 事件中设置它们的 Visible(可见)属性。

If the SideRuler or TopRuler=True：用户正将标尺标线拖进页面中，退出该过程，否则 PointerTool 会认为你正在画一个选定框。

CurrentTool：从 ToolButtons 控制对象中选定的当前按钮。

Case PointerTool：Find_Object 过程检查是否在某个对象的边界方框内单击了鼠标指针，如果单击了，此过程就显示出该对象的节点和图柄，高亮显示该对象。

BoundingBox_Down：此过程保存鼠标单击时的当前 X，Y 位置，绘图操作即从这些坐标开始。

Case BézierTool：如果一条 Bézier 曲线的图柄是可见的，则连接图柄和节点的点线也是可见的。如果用户单击了 ArtWork 桌面，就应该擦去点线，并隐藏 Bézier 图柄和节点。将点线标记为 False，再调用 BezUpdate 就可以完成该任务。

Case TextTool：文字放置在屏幕上的标题控制对象内。因为标签位于 Visual Basic 的中间图形层，不能放到图形对象的后面，所以必须将 Label.Caption(输入的文字)转换成图形类型。RedrawText 过程使用 Print 语句来达到此目的。为了将文字置于正确的文字光标位置，必须检索到所用字符的前导和文字高度。用 Visual Basic 的 TextHeight 方法不能得到内部前导值，因此应该使用 API TextMetric 函数。(参考第 12 章“文字对齐程序”有关文字属性的说明)。

Txt.Left 和 Txt.Top：文字在桌面上的位置。

最后，如果当前选定的工具是 Bézier 工具，检查一下 Bézier 节点是否可见，如果可见，就将这些节点改为不可见。

2.20 在桌面上拖曳或移动鼠标

MouseMove 事件执行所有的移动坐标功能。还必须搞清楚的是，是否确实正在绘制一个对象或从两个标尺中拖动标尺标线(参见清单 2.13)。

清单 2.13　ArtWork_Move 计算鼠标的坐标

```
Sub Artwork_MouseMove (Button%, Shift%, X As Single, Y As Single)
Xstr = Format $ (X \ PixelX, ″000″)
Ystr = Format $ (Y \ PixelY, ″000″)
XYlabel.Caption = ″[″ + Xstr + ″ , ″ + Ystr + ″]″
RulerGuides Index, Button, Shift, X, Y
  If SideRuler = True Or TopRuler = True Then
    GuideLines Index, Button, X, Y: Exit Sub
  End If
If Button = Left_Button Then
  BoundingBox_Move Shift, X, Y
  X2 = X: Y2 = Y
  BoundingBox_Move Shift, X, Y
End If
End Sub
```

X,Y 位置显示在 XYLabel 中,它位于靠近屏幕左下角的地方(参见图 2.2)。正在使用的是 twip 度量单位,因此,X,Y 除以 PixelX,PixelY 就可得到标题(caption)的象素单位数。这与使用 TwipsPerPixelX 和 TwipsPerPixelY 是一样的。

RulerGuides:该过程画出每个标尺上移动的线,给出鼠标的当前位置。如果单击了侧边的 Ruler(0)或顶部的 Ruler(1),则应调用 GuideLines 在页面上画出标线(点线)。拖动任何一个标尺上的标线都可以产生此效果。

BoundingBox_Move:启动当前绘图函数的图形语句,绘图函数有 Box,Circel 和 Line。绘制第一幅图象时 DrawMode 等于 Inverse(反转)。然后用 X2=X:Y2=Y 交换起始位置(MouseDown)X,Y,X,Y 就是当前鼠标移动的 X,Y 位置;不停地交换起始和终止点,每移动一次鼠标就画出两幅图象,这就意味着移动鼠标时会画出成千上万的边界方框。Inverse 属性擦去每次移动时所画的第二幅图象,这样从视觉上就看到一个拉伸的对象。注释掉(加上′)Inverse 语句。请记住,在每次交换值之前都要先画出对象,否则就会画出成千上万的不见边界的方框。

图 2.10 显示了一个拉伸的方框。

2.21　完成绘图

MouseUp 事件最后画出对象。如果单击了鼠标右键,就会弹出一个菜单,该菜单列出 Artisan 窗体主菜单的可选项(见清单 2.14)。

清单 2.14　ArtWork_MouseUp 在桌面上画出最终的对象

```
Sub ArtWork_MouseUp (Button%, Shift%, X As Single, Y As Single)
If CurrentTool <> 8 Then ArtWork.MousePointer = 0
If SideRuler = True Or TopRuler = True Then
   Ruler_MouseUp Index, Button, Shift, X, Y
   Exit Sub
```

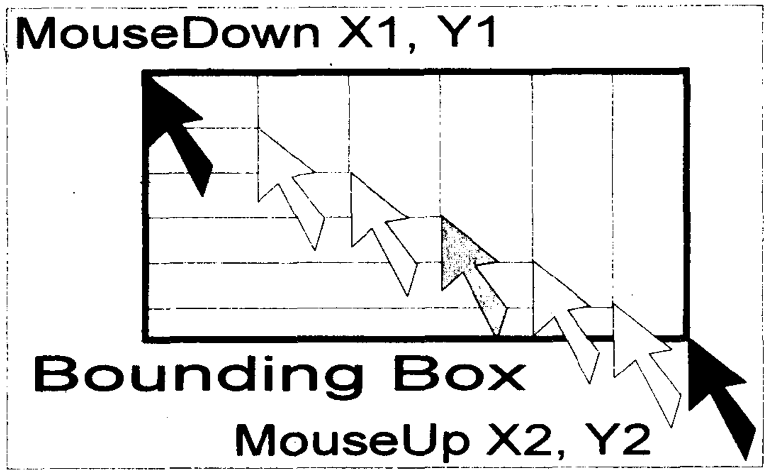


图 2.10　一个拉伸的方框

```
End If
If Button = Left_Button Then
  Select Case CurrentTool
     Case PointerTool
       If FlagMove = True Then UpdateObjects： FlagMove = False： Exit Sub
       If Node(0).Visible Then Handles： Exit Sub
       If BezNode1.Visible Then Handles： Exit Sub
         If MoveMouse() = True Then
           ArtWork.Line (X1， Y1)-(X2， Y2)， YELLOW， B
           FindAllObjects X1， Y1， X2， Y2， X， Y
         End If
  Case BézierTool
    If MoveMouse() = True Then
    ArtWork.Line (X1， Y1)-(X2， Y2)， YELLOW， B
    End If
  Case ZoomTool
       '''''''''''''''''''''''''
  Case LineTool To CircleTool
      If MoveMouse() = True Then
      BoundingBox_Up X1， Y1， X2， Y2
      If B = Total Then Draw = True： UpdateObjects
    End If
  Case TextTool
    Txt.BorderStyle = 1： txt.Width = 15： txt.Visible = True
  End Select
ArtWork.DrawWidth = 1： ArtWork.DrawStyle = Solid
End If
If Button = 2 And Menu_ArrangeSelection(3).Checked And CurrentTool < 8 Then
    Dim X_Y As POINTAPI：GetCursorPos X_Y
    R = TrackPopupMenu(GetSubMenu(GetMenu(hWnd)， LastMenu)，
        0， X_Y.X， X_Y.Y， 0， hWnd， 0)
ElseIf CurrentTool = TextTool And Button = 2 Then ArtWork.CurrentY.= txt.Top
    TextForm.Show Modal
    If txt.Caption > "" Then ReDrawText
End If
End Sub
```

如果 ArtWork 的鼠标指针改变了，就应该将其设回默认设置。

如果单击了侧边标尺或顶部标尺，则画出标线（点线），然后退出该子程序。

如果按了鼠标左键，继续执行 Select/Case 语句，否则就是按了鼠标右键，显示出一个弹出式菜单。

Case PointerTool：如果按下 Shift 键，就可能是在移动对象，调用 UpdateObjects 在新位置上画出该对象。关闭 FlagMove 标记，然后退出该过程，不执行以下的绘图语句。

如果一个标准节点是可见的，说明选定了某个对象，因此，不需要执行 UpdateObjects，就可以退出该子程序。

如果一个 Bézier 节点是可见的，说明完成了对一条曲线的操作，在曲线周围加上图柄，然后退出该子程序。

如果在任何方向上移动鼠标超过了 45 个 twip，则说明在某个对象周围画出了一个方框以选定它，调用 FindAllObjects 过程，查看方框区域内是否有什么对象。

Case BézierTool：与 Case PointerTool 相同，只不过没有调用 FindAllObjects，可以在这里加入对 Bézier 节点的调用，高亮显示方框内的节点。

Case ZoomTool：留待将来需要时使用。

Case LineTool to CircleTool：如果当前选定的是一个 Line（直线），Circle（圆）或 box（方框）工具，则改变 DrawMode，将它改为正常方式（Copy_Pen-13）。调用 BoundingBox_Up 过程在桌面画出最终对象。

如果 B 的值（当前对象）等于 Total（到目前为止所画的全部对象），就将标记变量 Draw 的值设为 True。UpdateObjects 就会跳过重画所有对象的 Loop（循环）语句，这样就会加快程序的运行速度。

附注：UpdateObjects 过程中每次都重画所有的对象会极度降低程序的执行速度，所要做的只是重画那些因为拉伸、缩放或移动而改变了的屏幕区域。Visual Basic 的剪切控制对象不适于这种要求，而且如果在执行某些调用时发生 AutoRedraw（自动重画），ArtWork_Paint 事件又可能会引起副作用，因此应该编写一段仅重画给定区域内对象的代码。

Case TextTool：用来显示文字插入点起始位置的是一条宽度为 15 个 twip 的竖线。该竖线由一个实心标题构成，它的 BorderStyle 设成可见（1）。输入文字时，其边界设成不可见（0），在继续输入过程中，该竖线不断移动，给你的感觉是直接在页面上输入，竖线的位置是用 Txt.Left 和 Txt.Top 的值来设定的，这两个值在 MouseDown 事件中定义。

ArtWork 的 DrawWidth 被复位为 1，因为任何时候都可以用 LineButtons 来改变该值。

ArtWork 的 DrawStyle 被重设为 0（实线），因为在画点线选定方框时可能改变此值。

如果单击了鼠标右键，就显示出一个弹出菜单。该过程通过 API 函数 TrackPopUpMenu 来处理。菜单弹出的位置由用户定义。

GetCursorPos：算出鼠标的当前位置。该 API 函数要求用户定义 Type 结构，并且所有的 X、Y 位置均是以屏幕象素为单位。

2.22 直线到 Bézier 曲线

如果双击 Bézier 曲线的一条线，并且当前工具是 Bézier 工具，则 Node Edit 窗体以模式(Modal)状态显示。

附注： *在模式(Modal)状态中继续执行之前，用户必须选定一个按钮。*

清单 2.15 ArtWork_DlbClick 打开一个节点编辑窗体，将一条直线转换成一条 Bézier 曲线

```
Sub Artwork_DblClick ()
If FlagLine(B) = B Then NodeEdit! To_Line.Enabled = False: [CI:CCC]
    NodeEdit! To_Curve.Enabled = True
If FlagBézier(B) = B Then NodeEdit! To_Curve.Enabled = False: [CI:CCC]
    NodeEdit! To_Line.Enabled = True
If CurrentTool = 1 Then NodeEdit.Show Modal Else Exit Sub
Select Case True
  Case Convert_Curve
    FlagLine(B) = 0: FlagBézier(B) = B
    BezErase = True: BezHandles
    BezHandle1.Visible = True: BezHandle2.Visible = True
    BezNode1.Visible = True: BezNode2.Visible = True
    Convert_Curve = False: NodeEdit! To_Line.Enabled = True
  Case Convert_Line
    FlagBézier(B) = 0: FlagLine(B) = B
    BezNode1.Visible = False: BezNode2.Visible = False
    W(B) = P1X(B): N(B) = P1Y(B)
    E(B) = P2X(B): S(B) = p2Y(B)
    Nodes
    Convert_Line = False: NodeEdit! To_Curve.Enabled = True
    UpdateObjects
End Select
End Sub
```

如果选定了按钮 To_Curve(该按钮将 Convert_Curve 标记设为 True)，则该直线的数组标记(FlagLine)数设为 0，标记为 0 的对象不能画出。Bézier 数组标记(FlagBézier)数被设成该直线的标记索引号。

然后 BezErase 被标记为 True，该直线被转换成一条曲线，其图柄和节点都在新曲线上，此过程是通过调用 BezHandle 来完成的。

所有 Bézier 图柄和节点都是不可见的，因此先将它们设为可见的，然后将 Convert_Curve 标记的值恢复成默认值 False。

最后，如果边框颜色是黑色，则将其值设置成当前画笔颜色。

2.23　在桌面上输入字符

当选定了文字工具并且光标在页面上时，就可以输入字符了。本程序中字符格式化或删除功能是很有限的，因为一个字处理程序本身就是一本书。不过还是应该通过 Artisan 程序来掌握基本的关于如何放置或对齐文字的方法。

还可以单击鼠标右键，显示出一个简单的字体选择对话框。

清单 2.16　ArtWork_KeyPress 统计在桌面上输入的文字的行数

```
Sub ArtWork_KeyPress (KeyAscii As Integer)
If CurrentTool = TextTool Then
   Select Case KeyAscii
     Case 13 'Return
     crCounter = crCounter + 1
     Case 8 'BackSpace
        If txt.Caption = "" Then Exit Sub
        If txt.Caption > "" Then
           If Mid $ (txt.Caption, Len(txt.Caption)) = Chr(13) [CI:CCC]
           Then crCounter = crCounter - 1
Txt.Caption = Mid $ (txt.Caption, 1, Len(txt.Caption) - 1): Exit Sub
  End If
End Select
   txt.Caption = txt.Caption + Chr(KeyAscii)
   StatusLabel.Caption = txt.FontName
   If txt.Caption > "" Then txt.BorderStyle = 0
End If
End Sub
```

ArtWork_KeyPress：如果使用文字工具，则文字先是输入到 Txt 标题框中。Txt 标题框控制对象是透明的，所以看上去好象是直接在桌面上输入的。crCounter 变量用来统计回车键的个数。它只是用来在保存和打开文件时抽取出每一行。如果采用更好的文件存储和打开过程（二进制或顺序的），这就用不着了。当敲了退格键时，就用 MID $ 语句擦除字符或回车字符。两个 Case 表达式（13 和 8）分别是回车键和退格键的 ASCII 值。

2.24　在桌面上重画文字

输入时显示在桌面上的文字实际上是在 Txt 标题框内，在完成输入后必须将这些文字传送到 ArtWork 桌面上（见清单 2.17）。

清单 2.17 ArtWork_LostFocus 重画文字

```
Sub ArtWork_LostFocus ()
If CurrentTool = TextTool And txt.Caption <> "" Then ReDrawText
End Sub
```

如果用户在 Txt 标题框中输入了文字并单击了别的控制对象，则该标题框就会隐去，并用 RedrawText 过程更新文字。

图 2.11 显示了一条转换成曲线的直线，Node Edit 窗体和 Txt 标题框控制对象。

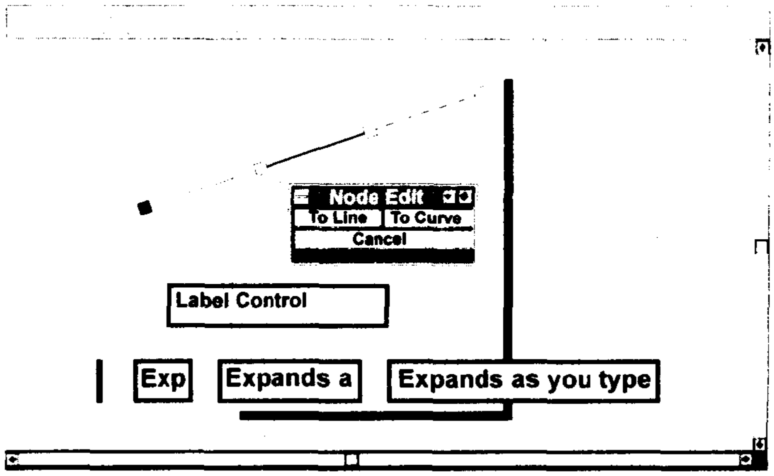


图 2.11 一条转换成了曲线的直线，Node Edit 窗体和 Txt 标题控制对象

2.25 BoundingBox_MouseDown 过程

BoundingBox_MouseDown 过程展示了怎样从一个边界方框的中心位置移动一个对象。这是在屏幕上移动对象的几种方法中的一种。其他的过程请参见第 1 章的 MoveTest 程序。

清单 2.18 BoundingBox_Down 在绘图之前存储一个对象的位置

```
Sub BoundingBox_Down (X As Single, Y As Single)
For R = -30 To 30 Step 15: For RR = -30 To 30 Step 15
If ArtWork.Point(X - R, Y - RR) <> ArtWork.BackColor Then
  FlagMove = True
   XCenter = X - ((W(B) \ 2) + (E(B)) \ 2)
   YCenter = Y - ((N(B) \ 2) + (S(B)) \ 2)
   OFFSETx = XCenter: OFFSETy = YCenter: Exit For
  End If
'ArtWork.PSet (X - R, Y - RR) 'use To view point area
If FlagMove = True Then Exit For
Next RR: Next R
  X1 = X: Y1 = Y: X2 = X1: Y2 = Y1
  Radius = 0
  ArtWork.DrawMode = Inverse
 XX1 = W(B)
 YY1 = N(B)
```

```
    XX2 = E(B)
    YY2 = S(B)
  End Sub
```

For R=－30 To 30 Step 15：仅当一个对象位于鼠标指针的两个象素之内时才能移动该对象。可以用 Visual Basic 的 Point 语句，递增 X，Y 的值以查找相应颜色的方法来计算出该距离值。嵌套的 For/Next 语句指向了这两个象素范围内的所有象素。如果没有搜索到 ArtWork 的 BackColor(白色)，就用两个 Exit For 语句跳出循环。

如果一个对象是用抖动颜色填充的，则某些抖动图案中彩色象素之间的空白可能会等于桌面的 BackColor(背景色)。这种情况下用 Point 语句就可能找不到对象。

FlagMove：说明 Point 语句找到了一种不是白色的颜色，将该标记置成 True 就可以在 BoundingBox_Move 过程中移动该对象。

XCenter 和 YCenter：计算对象边界方框的中心位置。

OFFSETx 和 OFFSETy：这些公式将对象置于鼠标指针的正中心位置。这两个位移值还在 MoveObjects 过程中用来将该对象重新移到原来的位置。

X1，Y1，X2 和 Y2：存储临时左上角和右下角的 X，Y 位置，绘制对象的图形语句中要用这些值。

半径(Radius)被重设为 0，这样在画一个新圆对象时不会与 Circle 语句相冲突，还可以避免发生运行期间错误。

ArtWork. DrawMode：设置为 Inverse，这样在发生 MouseMove 事件时就产生一种一次只画了一个对象的错觉。

XX2，YY2，XX1 和 YY1：临时存储单元，用来在用户使用 Handles 过程拉伸了一个边界方框，或将一个对象移到新的屏幕位置之后清除该对象的上一个位置。

MouseMove 函数

MouseMove 是一个判断在采取任何别的操作之前鼠标是否移动了预定距离的函数。如果鼠标从发生 MouseDown 事件的 X，Y 坐标处开始移动了给定数目的象素，则开始绘制一个对象。

如果用户碰巧单击了 ArtWork 桌面而实际上并不想画某个对象，那么该过程就可以终止任何无意的绘图动作。

2.26　在桌面上执行实际的绘图操作

拖曳鼠标时，Boundingbox_MouseMove 事件就会画出一个对象。Artisan 程序中的所有图形都只用了 Basic 语言中的图形语句。第 15 章中的 ArtAPI 程序重构了 Artisan 程序，它只用了 Windows 的 API 绘图函数。你会发现，混合使用这两种类型的绘图函数可以节省编程时间。

清单 2.19 BoundingBox_Move 画出实际的图形

```
Sub BoundingBox_Move (Shift As Integer, X As Single, Y As Single)
If CurrentTool = 0 And FlagMove = True And [CI:CCC]
    MoveMouse() = True Then MoveObject Button, [CI:CCC]
        Shift, X, Y: Exit Sub
If Node(0). Visible = True Then Exit Sub
If MoveMouse() = True Then
  Select Case CurrentTool
    Case PointerTool To BézierTool
        ArtWork. DrawMode = Xor_Pen
        ArtWork. DrawStyle = Dot
        ArtWork. Line (X1, Y1)-(X2, Y2), YELLOW, B
    Case LineTool
        ArtWork. Line (X1, Y1)-(X2, Y2)
    Case BoxTool
        ArtWork. Line (X1, Y1)-(X2, Y2), , B
    Case CircleTool
        LastCircleGood = PerfectCircle(X1, Y1, X2, Y2)
        ArtWork. Circle (XCenter, YCenter), Radius, , , , Aspect
  End Select
End If
End Sub
```

If CurrentTool＝0：如果当前选定的工具是指针工具，而且 FlagMove 变量在 BoundingBox_Down 过程中置成了 True，那么就应该移动一个对象而不是画一个对象。该情况下，应该退出本过程。

If Node(0). Visible：有一个标准节点是可见的，这意味着某个对象被高亮显示，因此不能绘制对象，只能退出本过程。

Select/Case：根据当前选定的工具，执行下面的 Case 表达式之一：

Case PointerTool To BézierTool：用以在一个对象周围绘制一个选定方框。将 DrawMode 改为 Xor_Pen(7)。选定方框就是一个边线是点线的方框，所以，应该用 ArtWork. DrawStyle＝Dot 将绘图式样改为点线。

该 Line 语句生成一个边线为黄色点线的方框。在 Xor(异或)模式下设定黄色时，得到的边线颜色是青色(淡蓝色)的。当对象经过别的颜色(如一个用绿色填充的方框)时，该区域内的线就变成黄色。在选定方框经过之后，背景颜色仍保持其原来的颜色(本例中是绿色)。

Case LineTool：该语句删去了 B 选项，因此只画一条直线。

Case BoxTool：该语句加上了 B 选项，因此画一个方框。

Case CircleTool：LastGoodCircle：调用 PerfectCircle 过程，该过程与在 Visual Basic 的 IconWorks 例子中看到的一样，它返回 Circle 语句中所需的 XCenter，YCenter 的值。不用 PerfectCircle 函数，API ellipse 调用也可以完成同样的功能。

2.27 给新对象指定一个引用号

此处给所有新绘制的对象设定索引号。变量 B 可控制赋予对象属性值的种类。可以指

定给一个新对象的属性有：对象边框的颜色、边框宽度、填充颜色以及在屏幕上的位置，当然，这只是其中的部分属性。

B 的值代表当前对象的值。该值可用来在几个索引数组的任一个中存储属性值。

UpdateObjects 过程集中了所有的属性(其索引值就是对象的编号)并绘出最终的对象。

清单 2.20　BoundingBox_Up 设定图形的对象索引号

```
Sub BoundingBox_Up (X1, Y1, X2, Y2)
If MoveMouse() = True Then
  Total = Total + 1: B = Total
  StatusLabel.Caption = Format$ (B)
  LineWidth(B) = ArtWork.DrawWidth
  W(B) = X1: E(B) = X2: N(B) = Y1: S(B) = Y2
    Select Case CurrentTool
      Case PointerTool
      Exit Sub
      Case BézierTool
      Exit Sub
      Case ZoomTool
      Exit Sub
      Case LineTool
      FlagLine(B) = B
      Nodes
      Case BoxTool
      FlagBox(B) = B
      FillValue(B) = 1
      Nodes
      Case CircleTool
      FlagCircle(B) = B
      FillValue(B) = 1
      Nodes
      Case TextTool
      ' no code needed here
  End Select
End If
End Sub
```

这是本应用程序中最重要的过程。在每一个对象画出后，其边界方框就被指定了一个编号。该编号被赋予变量 B。B 就控制着可能赋予该对象的每一项属性(如方框，Bézier 曲线或文字)。

窗体装入时，B 的值等于 0，所画对象的总数为 0，页面上是空白的。total 变量记录所创建的所有对象。

B 告诉你哪一个边界方框(对象)被高亮显示，并指定了每种属性数组中(诸如标记 flag，式样 Style 或颜色 color)的索引号。例如，如果所画的第 3 个对象是一个圆，B 就等于 3，并且边框宽度、颜色、填充颜色、边界方框位置在每个属性数组中的索引均为 3。

If MoveMouse＝True：说明已移动鼠标足够远，起动了绘图事件，否则说明只是在页面上单击了一下鼠标。

Total＝Total＋1：对象计数增 1。Total 等于到目前为止所画对象的总数。

B＝Total：给边界方框(对象)指定一个编号。

LineWidth(B)：该 LineWidth 索引项等于当前的 DrawWidth。因为画图宽度不会为 0，所以 LineWidth(B)的值可以防止发生运行期间错误。

W(B)、N(B)、E(B)和 S(B)：边界方框的角点坐标。X1、Y1、X2、Y2 只用作临时的坐标保存单元。给这些变量指定 B 值就会告知程序这些角点位置属于哪个边界方框。

附注： W、N、E、S 变量分别代表边界方框的西、北、东、南点。有些 API 调用使用一种 Rectangle 类型结构，该结构采用的变量是 left(左)、right(右)、top(上)、bottom(下)。因此，下面的逻辑引用都采用 W、E、N、S，比较容易输入，也易于记忆。

Select Case CurrentTool：这取决于使用了下面的哪一个工具。

Case PointerTool：用该工具不能绘制对象，因此退出该子程序。

Case BézierTool：Bézier 曲线的绘制是用 DrawBézier 过程来处理的，并且其边界方框是由 BézierStore 过程设定的，退出该子程序。

Case ZoomTool：该工具留待将来使用，设有相应的代码，所以退出该子程序。

Case LineTool：在 FlagLine 索引数组中插入一个对象编号(B)，因此程序知道该边界方框内所包含的是一条直线。调用 Nodes 在该直线的两端设置两个节点。

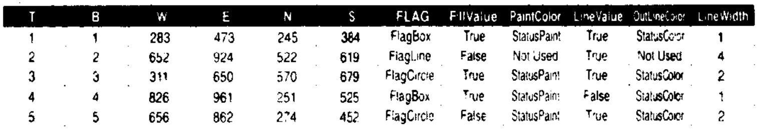
Case BoxTool：在 FlagBox 索引数组中插入一个对象编号(B)，因此，程序知道该边界方框内所包含的是一个方框；还在 FillValue 索引数组中插入一个对象编号(B)，程序就会知道该方框是实心(0)的还是透明的(1)。它也调用 Nodes 在该方框的每个角上设置一个节点。

Case CircleTool：在 FlagCircle 索引数组中插入一个对象编号(B)，因此程序知道该边界方框内所包含的是一个圆；还在 FillValue 索引数组中插入一个对象编号(B)，这样，程序就会知道该圆是实心的(0)还是透明的(1)。它还调用 Nodes 在该圆的 12 点钟位置设置一个节点。

Case TextTool：文字的绘制是由 DrawText 过程处理的，因此可以退出此子程序。

附注： 每一个图表标题(见图 2.12)代表了一个有索引的数组，但 B 和 T 除外。变量 B 代表当前选定或最后绘制的对象。变量 T 代表至今为止所画的对象的总数。如果所画的第 3 个对象(B＝3)是一个方框，则与该方框关联的所有图表标题的索引都是 3。UpdateObjects 调用就可以用这些存储好的值重建对象，并指向正确的索引号。

已经绘制的对象的例子请参见图 2.12。



T	B	W	E	N	S	FLAG	FillValue	PaintColor	LineValue	OutLineColor	LineWidth
1	1	283	473	245	384	FlagBox	True	StatusPaint	True	StatusColor	1
2	2	652	924	522	619	FlagLine	False	Not Used	True	Not Used	4
3	3	311	650	570	679	FlagCircle	True	StatusPaint	True	StatusColor	2
4	4	826	961	251	525	FlagBox	True	StatusPaint	False	StatusColor	1
5	5	656	862	274	452	FlagCircle	False	StatusPaint	True	StatusColor	2

图 2.12　至今为止已绘制的 5 个对象

2.28　本章小结

现在已经知道了 ArtWork 的各种事件是怎样与 BoundingBox 过程协同工作，从而在桌面上绘制一个对象的。不过并没有到此结束，接下去的三章将会完成对 Artisan 绘图程序的讨论。每一章给你介绍一种新功能。所有这些功能都只使用了 Basic 语言的语句。

每一个事件和函数中的代码都设计得很长，本书避免包含奇怪的循环、嵌套循环等等。这将给你一个机会，让你理解每个过程的结构，并在将来对这些代码进行压缩。

第 15 章中的 ArtAPI 程序（使用了 Windows 的 API 图形函数）是 Artisan 程序的一个翻版，它包含了一些从下面三章中比较长的过程压缩而得的过程。不过，你还是不应该跳过任何一章而直接转到第 15 章，理由是，ArtAPI 程序在某些部分变得相当复杂。尤其是当将直线和 Bézier 曲线连接起来并编辑它们时更是如此。

第 3 章 Bézier 曲线

Bézier 曲线是一种沿着给定的轨迹绘制柔性曲线的算法，该算法是根据法国一位数学家 Pierre Bézier 命名的，它是作为雷诺汽车公司的 CAD(Computer-Aided Design，计算机辅助设计)系统的一部分而开发的。目前大多数图形设计用的计算机和打印机语言程序都采用了 Bézier 曲线公式。

Bézier 曲线点可通过几种方法来构造。每一条曲线都有一个起点和一个终点，本章中分别称为 P1 和 P2，每条曲线也都至少有一个或更多的控制图柄。一条 3 个控制点的 Bézier 曲线通常叫做 B 样条曲线，它用了控制点 P1，P2 和一个控制图柄。本书所采用的这种 Bézier 曲线有 4 个控制点，包括曲线的起点(P1)，终点(P2)和两个控制图柄点(H1 和 H2)。这种类型的 Bézier 曲线通常叫做 4 点 Bézier 样条曲线。图 3.1 给出了 Bézier 曲线的这些点。

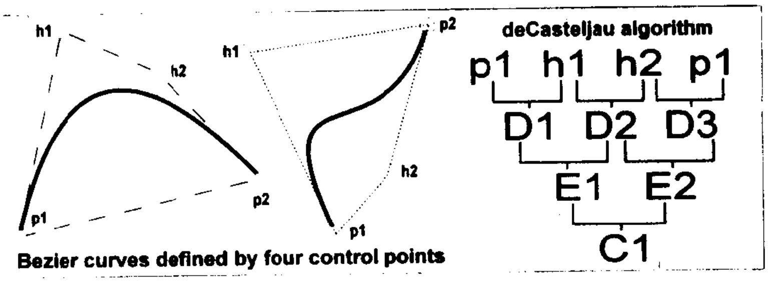

h1
h2
p1
p2
h1
p2
h2
p1
Bezier curves defined by four control points
deCasteljau algorithm
p1 h1 h2 p1
D1 D2 D3
E1 E2
C1

图 3.1 deCasteljau 划分公式

下面给出计算 4 点 Bézier 样条曲线的公式：

$$P(u)=(1-u)^3P1+3u(1-u)^2H1+3u^2(1-u)H2+u^3p2$$

可以用 deCasteljau 算法简化 Bézier 曲线的计算。用整数反复除 Bézier 曲线的 4 个控制点(P1，H1，H2，P2)，就可以求出所有曲线段的端点。从点(P1)开始，用直线连接所求出的每一点，最后一点就是 P2。曲线的轨迹取决于 H1，H2 的位置，H1，H2 叫做曲线控制图柄。

deCasteljau 算法是一种简单的划分方法，它看上去像是一个树图。可以把曲线的起点(P1)看作总是与 1 号(H1)控制图柄有关，曲线的终点与第 2 号(H2)控制图柄相关。P1 和 H1 控制点是曲线的左边，P2 和 H2 控制点是曲线的右边。下面的例子中，用该算法求出了曲线左边控制点，两个曲线控制图柄及曲线右边控制点之间的中间点，然后继续计算这些中间点之间的中间点，直至求出划分点(C1)，该点真正位于曲线上。

下列公式计算出曲线控制点之间的中点以及这些中间点之间的中点：

$(P1+H1)/2=D1$

$(H1+H2)/2=D2$

$(P2+H2)/2=D3$

$(D1+D2)/2=E1$

$(D2+D3)/2=E2$

(E1+E2)/2=C1(该点在曲线上)

图 3.2 给出了求一条 Bézier 曲线中间划分点(C1)的第一个步骤。四个 Bézier 控制点是 P1,H1,H2 和 P2,设想一个由这四个控制点连接而成的多边形形状,不管曲线的形状如何,它总是在这个想象的四角框内。

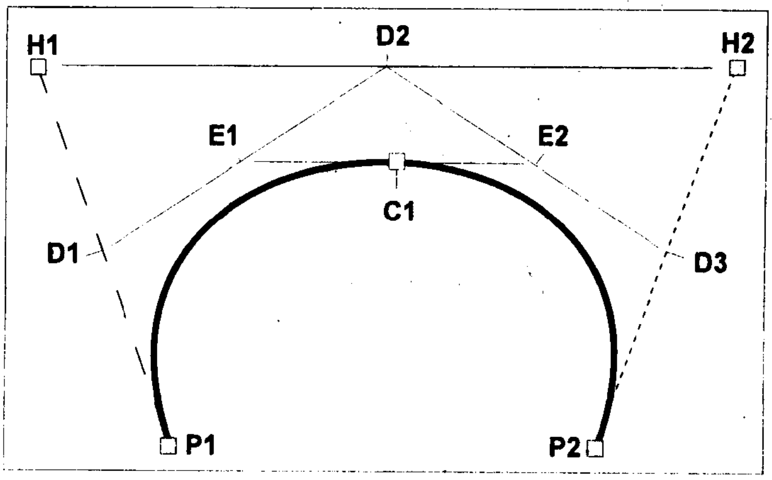

H1
D2
H2
E1
E2
C1
D1
D3
P1
P2

图 3.2　Bezier 曲线的一个划分点

从 P1 和 H1 点开始,使用 deCasteljau 算法,将这两个点的位置加起来再除以 2 就得到了它们的中间点(D1):

D1=(P1+H1)/2

然后再将 H1 和 H2 加起来除以 2 就得到了 D2。

继续运用该公式直至求出曲线的分划点(C1)。第一个 C1 的位置就是要求的曲线中点。

图 3.3 给出了曲线的中心划分点(C1)。若要搞清怎样计算一个新划分点,就必须把该曲线分成两半来看。

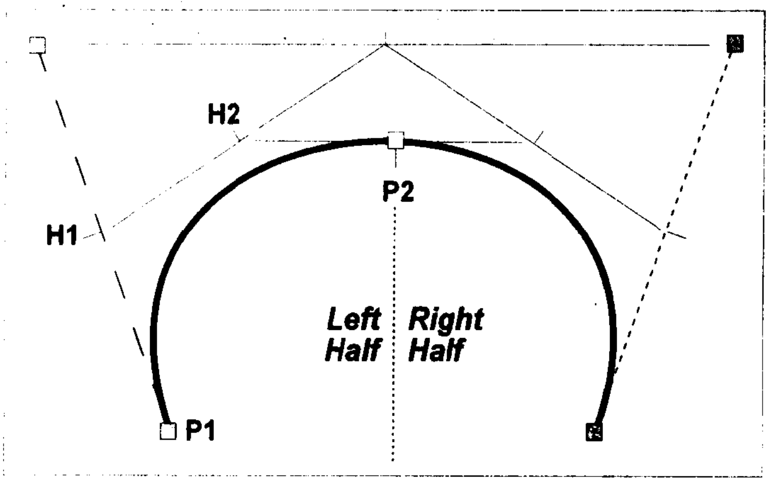

H2
P2
H1
Left Half
Right Half
P1

图 3.3　Bézier 曲线的左右两半

将 C1 点的位置存在一个数组中,将其改名为 P2,如果你将此图顺时针方向旋转 45 度,

你就会明白问题所在。

假设中心点(C1)就是曲线的终点(P2),D1 和 E1 就是两个 Bézier 图柄点(H1 和 H2)。

图 3.4 利用了与图 3.2 中相同的公式,求出曲线左半部分的中心划分点。轻轻地沿顺时针方向旋转图 3.4,P1 和 C1 就变成了新的曲线起点和终点,重复图 3.2 中的步骤,就出现了一个“涡流”,随着不断逼近控制点 P1,此“涡流”越来越小。每一个新的 C1 点都落在曲线上。如果先顺时针再逆时针方向进行,就可以得到一个递归的公式,重复处理曲线的每一个部分。

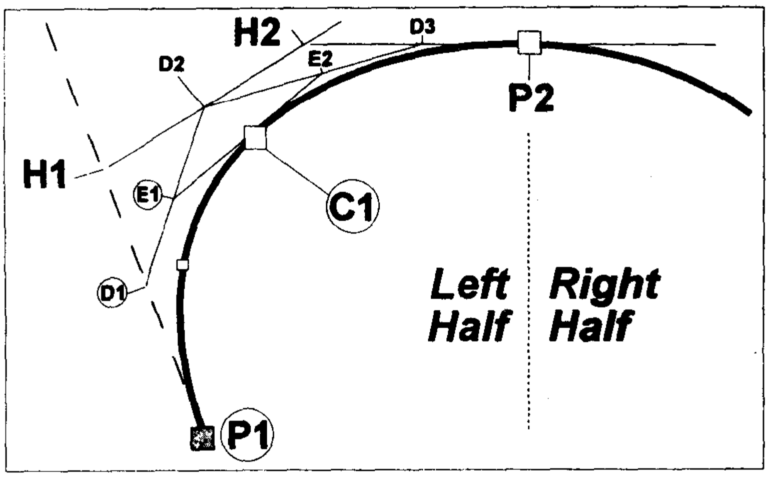

H2
D3
D2
E2
P2
H1
E1
C1
D1
Left Half
Right Half
P1

图 3.4 算出下一个曲线点

该 Bézier 曲线要画 32 个点。每一个 X,Y 点都用一种用户定义的数据类型 POINTAPI 存储,该结构有两个子标题 X 和 Y。(POINTAPI 类型在 ARTISAN7.BAS 文件中)。

每条曲线都有一个初始的起始点和终止点,节点与这两个点相连,因此 34 个点构成一条曲线。点 P1 是该曲线的起点,P2 是该曲线的终点。图 3.5 显示了一条 Bézier 曲线左边的曲线点,图 3.6 则显示了 Bézier 曲线右边的曲线点。

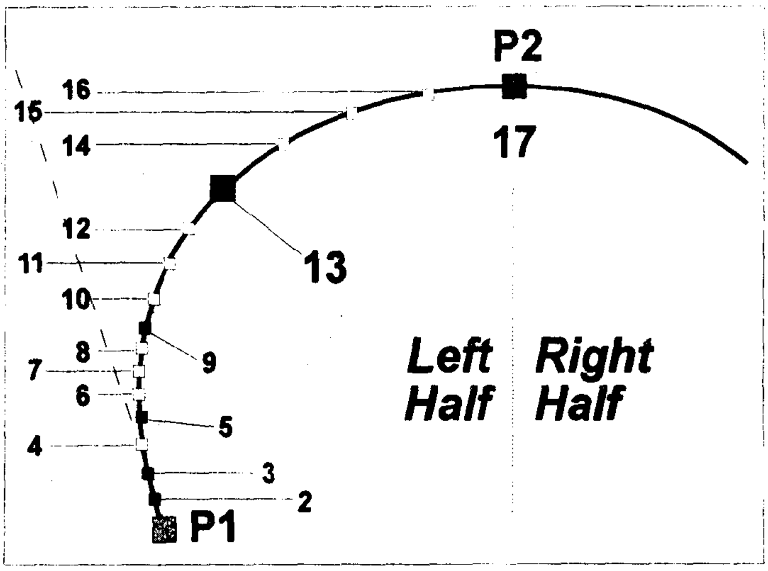

P2
16
15
14
17
12
11
13
10
8
9
7
6
5
4
3
2
P1
Left Half
Right Half

图 3.5 一条预绘制的(用户定义的)Bézier 曲线的左半部分

曲线点 17 对应着曲线中心点的 X，Y 位置，因为 34/2＝17。

Bediride 过程包含了一个 deCasteljau 算法公式。紧接着每个 BézierDivide 调用的四个值叫做参数(argument)。BézierDivide 的参数的顺序如下所示：

```
Sub BézierDivide(p1 As PointApi,h1 As PointApi,h2 As PointApi,p2 as PointApi)
```

这使得 BézierDivide 子程序认为传给它的四个参数就是 Bézier 节点 P1 和 P2 以及图柄点 H1 和 H2。

BezDraw 过程(如清单 3.1 所示)将两个图柄和新的控制点传给 BezDivide 过程中的除法公式，BezDivide 过程见清单 3.2，它算出每一个曲线点。最后的计算结果 C1(P).X 和 C1(P).Y 就构成实际的曲线点。

最后，BezDraw 过程取得所有 32 个点(这些点存储在数据数组 C1 中)，并用 Line 语句(或 Polyline)将这些点连起来就得到了一条看上去平滑的曲线。

图 3.6 中，曲线的右半部分是以曲线中心点为起点绘制的，该中心点就是数组 C1 中索引号为 17 的点。

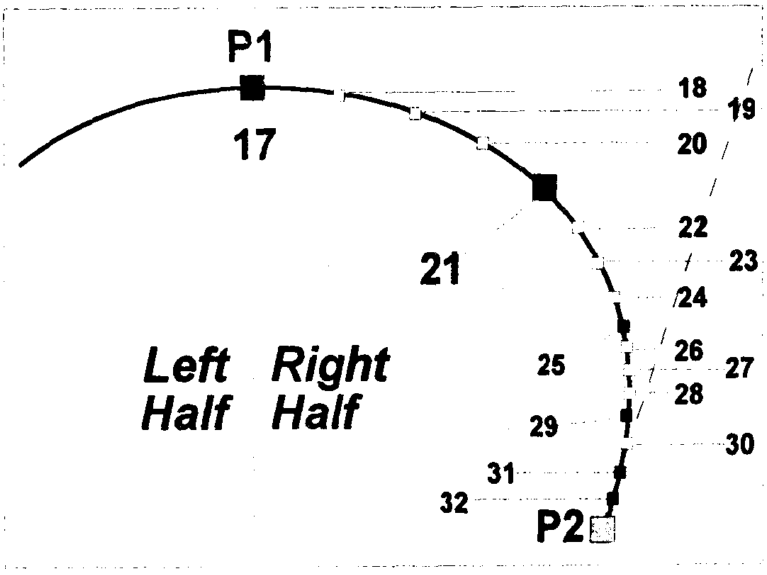


图 3.6　一条预绘制的 Bézier 曲线的右半部分

在曲线的左边，从点 2 到点 16 是计算出来的，它们存在于数据数组 C1 中，它们的曲线位置号就是其索引号。

在曲线的右边，从点 18 到点 32 是计算得来的，它们存在数组 C1 中，它们的位置点就是在数组中的索引号。Bézier 曲线的右边用其右边分划值来画点。请注意，这不是一个真正的在大多数 C 语言过程中所看到的那样的递归分划过程。画图方法都是预先计划好的，每一个点的计算顺序也是计划好的。

附注： 预绘制 Bézier 曲线的优点是代码运行很快。因为递归过程在计算一条光滑曲线时会使曲线点数翻倍。

不利之处是必须逐个计算每一个点，在过程中额外增加了 32 行代码。

最后生成的曲线(C1)有如下一个数据数组：

C1(2),C1(3),C1(4),C1(5),C1(6),C1(7),……直至……C1(32)

每一个 C1 索引号内都存储了两个值:X 和 Y,也就是 POINTAPI。

每一个 C1 索引的 X,Y 坐标如下所示:

C1(2).X,C1(2).Y—C1(3).X,C1(3).Y,……直至……C1(32).X,C1(32).Y

C1(index).X 保存指定曲线点的 X 坐标。

C1(index).Y 保存指定曲线点的 Y 坐标。

3.1 绘制 Bézier 曲线点

下面解释在 BezDraw 过程中用来绘制 Bézier 曲线的各个变量。

变量 P 后面的数字代表用来存储 deCasteljau 算法生成的每一个值的索引号。

P 还代表了给所绘制点指定的编号,这些点构成了曲线。从曲线的起点(P1)开始,到曲线的终点(P2)结束,曲线绘制点编号从 2 到 32,这样编号为 17 的曲线点就是曲线的中点。

首先用同样的公式算出曲线的左边,得到点 2 至点 16。

接着计算曲线的右边,基本上是左半部分的镜象变换,得到点 18 至点 32。清单 3.1 给出了曲线点的计算过程。

清单 3.1 计算预选定义参数的曲线点

```
Sub BezDraw ()
'''Left side of curve
p = 17: BézierDivide p1, h1, h2, p2
p = 13: BézierDivide p1, D1(17), E1(17), C1(17)
p = 9: BézierDivide p1, D1(13), E1(13), C1(13)
p = 5: BézierDivide p1, D1(9), E1(9), C1(9)
p = 3: BézierDivide p1, D1(5), E1(5), C1(5)
p = 2: BézierDivide p1, D1(3), E1(3), C1(3)
'''''
p = 4: BézierDivide C1(3), E2(3), D3(3), C1(5)
p = 7: BézierDivide C1(5), E2(5), D3(5), C1(9)
p = 6: BézierDivide C1(7), D1(7), E1(7), C1(5)
p = 8: BézierDivide C1(7), E2(7), D3(7), C1(9)
p = 11: BézierDivide C1(9), E2(9), D3(9), C1(13)
p = 10: BézierDivide C1(11), D1(11), E1(11), C1(9)
p = 12: BézierDivide C1(11), E2(11), D3(11), C1(13)
p = 15: BézierDivide C1(13), E2(13), D3(13), C1(17)
p = 14: BézierDivide C1(15), D1(15), E1(15), C1(13)
p = 16: BézierDivide C1(15), E2(15), D3(15), C1(17)
'''Rightside of curve
p = 21: BézierDivide C1(17), E2(17), D3(17), p2
p = 25: BézierDivide C1(21), E2(21), D3(21), p2
p = 29: BézierDivide C1(25), E2(25), D3(25), p2
p = 31: BézierDivide C1(29), E2(29), D3(29), p2
p = 32: BézierDivide C1(31), E2(31), D3(31), p2
'''''
p = 30: BézierDivide C1(29), D1(31), E1(31), C1(31)
p = 27: BézierDivide C1(25), D1(29), E1(29), C1(29)
p = 28: BézierDivide C1(29), E2(27), D3(27), C1(27)
```

```
p = 26: BézierDivide C1(25), D1(27), E1(27), C1(27)
p = 23: BézierDivide C1(21), D1(25), E1(25), C1(25)
p = 24: BézierDivide C1(25), E2(23), D3(23), C1(23)
p = 22: BézierDivide C1(21), D1(23), E1(23), C1(23)
p = 19: BézierDivide C1(17), D1(21), E1(21), C1(21)
p = 20: BézierDivide C1(21), E2(19), D3(19), C1(19)
p = 18: BézierDivide C1(17), D1(19), E1(19), C1(19)
If FlagPrinter = 2 Then GoTo PrintToPage
If ArtWork.AutoRedraw = False Then ArtWork.DrawMode = Inverse
ArtWork.DrawStyle = Solid
'P_Index(1).X = p1.X \ PIXELX
'P_Index(1).Y = p1.Y \ PIXELY
    'For P = 2 To 32
     'P_Index(P).X = C1(P).X \ PIXELX
     'P_Index(P).Y = C1(P).Y \ PIXELY
  'Next P
'P_Index(33).X = p2.X \ PIXELX
'P_Index(33).Y = p2.Y \ PIXELY
'NUM_PTS = 33
'R = Polyline(ArtWork.hDC, P_Index(1), NUM_PTS)
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
ArtWork.PSet (p1.X, p1.Y)
    For p = 2 To 32
        ArtWork.Line -(C1(p).X, C1(p).Y)
    Next p
ArtWork.Line -(p2.X, p2.Y)
Exit Sub
PrintToPage:
Printer.PSet (p1.X, p1.Y)
    For p = 2 To 32
        Printer.Line -(C1(p).X, C1(p).Y)
    Next p
Printer.Line -(p2.X, p2.Y)
Exit Sub
End Sub
```

曲线左侧:p 是一个临时整数、它传递数组 D1,D2,D3,E1,E2 和 C1 的索引号,它还代表点在曲线上的顺序,从 BezNode1(P1)开始,计数 32 次直至最后与 33 点(也就是 BezNode2(P2))相连接为止。

你使得 BézierDivide 认为你传给它的四个参数就是 Bézier 节点 P1,P2 和 Bézier 图柄点 H1,H2。

第一块计算代码分划出曲线左边的曲线点,如图 3.7 所示(黑色的节点从 17 开始)。这些曲线点用 C1(索引号)保存。因此头六个点存储在 C1(17),C1(13),C1(9),C1(5),C1(3)和 C1(2)中。

现在计算左边剩下的 10 个点,计算方法是算出从点 3 和点 5 开始的黑色节点之间的中间点,反方向算回来(见图 3.7 中的灰色节点)。如果在灰色和黑色节点之间还有间隙,就再计算出这些点之间的中间点并画成另一种点(图 3.7 中的白色节点)。

现在请看从 P=4 开始的第二个语句块。注意,第一个参数 C1 和第四个参数(还是 C1)

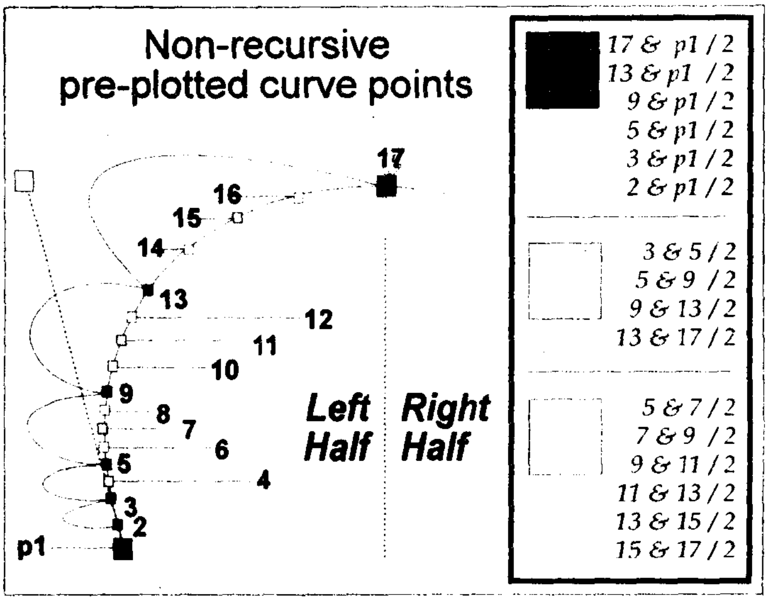

Non-recursive
pre-plotted curve points
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
p1
Left Half
Right Half
17 & p1 / 2
13 & p1 / 2
9 & p1 / 2
5 & p1 / 2
3 & p1 / 2
2 & p1 / 2
3 & 5 / 2
5 & 9 / 2
9 & 13 / 2
13 & 17 / 2
5 & 7 / 2
7 & 9 / 2
9 & 11 / 2
11 & 13 / 2
13 & 15 / 2
15 & 17 / 2

图 3.7 曲线点的计算顺序

被假设成两个 Bézier 节点。中间两个参数被假设成 Bézier 控制图柄。你正在做的就是生成一个“涡流”，该“涡流”与生成第一个点（即点 17）时相同，并逆时针旋转以构成曲线的左半部分。曲线的右半部分也是通过重复这种“涡流”图案生成的，不同的是从点 17 开始，顺时针方向旋转。随着不断逼近终点，每一次重复的图案就越来越小。P1 是左边曲线的起点，P2 是右边曲线的终点。

现在就可以在桌面上显示出该 Bézier 曲线了。有两种方法可以画出该 Bézier 曲线。如果想用 API 的 PolyLine 函数的话，就请注释掉当前使用 Basic Line 语句的五行语句并删去 API PolyLine 部分的注释符号。

PIXELX，PIXELY：API 的 PolyLine 只能按象素模式工作，所以将 twip 转换成象素，PIXELX 和 PIXELY 在 Form_Load 事件中定义。

P_Index：这是一个临时数组，它存放由 POINTS 类型结构定义的 X，Y 坐标，POINTS 类型结构在 ARTISAN7.BAS 文件中。

P1.X，P1.Y：（BezNode1）曲线的起点。

For P=2 to 32：将 P 设为起始值 2，计数至 32 结束。

P_Index(P).X=C1(P).X\TPX：将每一个 C1（曲线点）的 X 坐标存在数组 P_Index 中，X 点是按 twip 度量的，因此将它们除以 TwipsPerPixelX（TPX）转换成象素数。

P_Index(P).Y=C1(P).Y\TPY：将每一个 C1（曲线点）的 Y 坐标存在数组 P_Index 中，Y 点是以 twip 为单位的，因此将它们除以 TwipsPerPixelY（TPY）转换成象素数。

Next P：转回 For P 语句，P 值增 1，重复前面的各行代码，直至全部 32 个 X，Y 点都存进了 P_Index 为止。

P2.X 和 P2.Y：（BezNode2）曲线的终点。

NUM_PTS=33：数组索引大小。告诉 API 要连接的点数。

R=Polyline(ArtWork.hDC，P_Index(1)，NUM_PTS)：调用 API 函数 Polyline 并让它

执行下列操作：

在 ArtWork 图片框中画图
从存储在 P_Index 中索引号为 1 的 X,Y 点开始画图
当到达索引号 33 时停止画图

Line 语句：该语句与 API PolyLine 语句一样快，但键入的字符比较少。

ArtWork. Pset：通过用 Pset 语句设定 CurrentX，CurrentY 的方法设定 Line 语句的起始点 P1（BezNode1）。

For. . . Next 语句：用 C1 数组中的值继续画线。

ArtWork. Line：P2. X，P2. Y（BezNode2）是曲线的终点。

API Polyline 函数画出一系列线。这些线从第一点开始画，直到指定的最后一点。若要使用该函数，就必须在某个模块（.BAS 文件）中创建一个用户定义的点结构，点结构的形式如下：

```
Type POINTAPI
    X As Integer
    Y As Integer
End Type
```

可以将每一条线段的 X，Y 点存放在 POINTAPI 类型的变量中，如下所示：

```
Dim Pt(1 to N) As POINTAPI
Pt(1).X=154:Pt(1).Y=276
Pt(2).X=212:Pt(2).Y=280
```

下面是一个例子：

```
Polyline(Pic.Hdc,Pt(1),2)
```

上面一行代码的语法是这样的：

Pic. Hdc 是将要在其上画图的控制对象，Pt(1)是该直线的起点，数字 2 则是要画的点数。

3.2 deCasteljau 分划公式

清单 3.2 中使用长除法，将结果强制转换成整数（如图 3.8 所示）。

清单 3.2 deCasteljau 公式

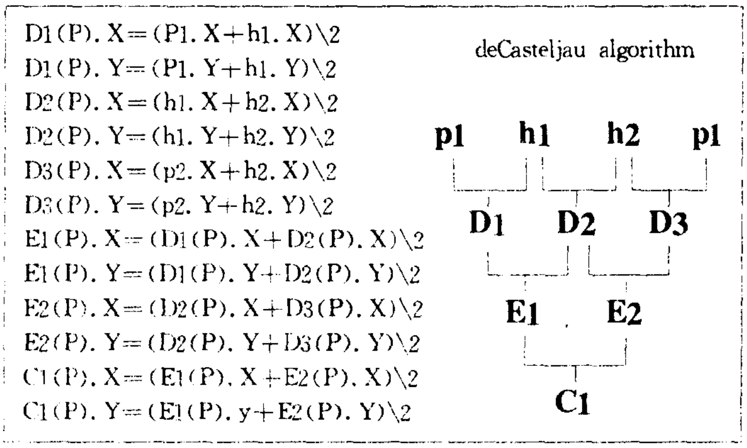
```
Sub BézierDivide (p1 As POINTS, h1 As POINTS, h2 As POINTS, p2 As POINTS)
D1(p).X = (p1.X \ 2 + h1.X \ 2)
D1(p).Y = (p1.Y \ 2 + h1.Y \ 2)
D2(p).X = (h1.X \ 2 + h2.X \ 2)
D2(p).Y = (h1.Y \ 2 + h2.Y \ 2)
D3(p).X = (p2.X \ 2 + h2.X \ 2)
D3(p).Y = (p2.Y \ 2 + h2.Y \ 2)
E1(p).X = (D1(p).X \ 2 + D2(p).X \ 2)
E1(p).Y = (D1(p).Y \ 2 + D2(p).Y \ 2)
E2(p).X = (D2(p).X \ 2 + D3(p).X \ 2)
```

```
E2(p).Y = (D2(p).Y \ 2 + D3(p).Y \ 2)
C1(p).X = (E1(p).X \ 2 + E2(p).X \ 2)
C1(p).Y = (E1(p).Y \ 2 + E2(p).Y \ 2)
'ArtWork.DrawWidth = 4
'ArtWork.PSet (C1(p).X, C1(p).Y)
'ArtWork.DrawWidth = 1
End Sub
```

p1.X\2+h1.X\2 可以简化为(p1.X+h1.X)\2，在象素模式下该公式运行得很好，但是在 twip 模式下，p1.x+h1.x 就有可能超过一个整数所能存储的最大数(有符号的32767)。

运用 deCasteljau 公式时总是由"\"运算符来做除法，"\"运算符将两个数相除，返回一个整数结果。如果使用"/"运算符做除法，执行速度会大大降低，因为"/"运算符将两个数相除返回一个浮点数(小数)结果。当使用 deCasteljau 公式时，浮点结果要求计算机处理器进行一些不必要的数学计算。

deCasteljau 算法的树结构请参见图 3.8。



```
D1(P).X=(P1.X+h1.X)\2
D1(P).Y=(P1.Y+h1.Y)\2
D2(P).X=(h1.X+h2.X)\2
D2(P).Y=(h1.Y+h2.Y)\2
D3(P).X=(p2.X+h2.X)\2
D3(P).Y=(p2.Y+h2.Y)\2
E1(P).X=(D1(P).X+D2(P).X)\2
E1(P).Y=(D1(P).Y+D2(P).Y)\2
E2(P).X=(D2(P).X+D3(P).X)\2
E2(P).Y=(D2(P).Y+D3(P).Y)\2
C1(P).X=(E1(P).X+E2(P).X)\2
C1(P).Y=(E1(P).y+E2(P).Y)\2
```

图 3.8 deCasteljau 算法的树结构

3.3 通过图柄拉伸曲线

BezHandle1_MouseDown 事件(清单 3.3)启动曲线的重画过程，使得在曲线改变形状时有一种移动的感觉。

清单 3.3 MouseDown 事件用图柄 1 画 Bézier 曲线

```
Sub BezHandle1_MouseDown (Button As Integer, Shift As Integer, X!, Y!)
XX2 = E(B): YY2 = S(B): XX1 = W(B): YY1 = N(B)
BezHandle1.Visible = False
BezNode1.BackColor = Blue: BezNode2.BackColor = Black
    BezMove
    BezDraw
End Sub
```

XX2,YY2,XX1,YY1：临时存放边界方框的大小，使得可以擦除前一个位置上的曲线。

BezHandle1：首先使当前 Bézier 图柄不可见，AutoRedraw 可能将鼠标残块（象素行残点）留在屏幕上。Bézier 节点和图柄是真正的图片框控制对象。连接 Bézier 节点和图柄的点线位于每个图片框的中央。线是用 Xor 模式绘制的，将这些点线与图片框交迭就会产生难以预料的颜色。

BezNode1. BackColor：改变节点的颜色仅仅是为了在视觉上引起注意。

BezMove：调用 BezMove 过程画出第一条曲线。BezMove 按正常模式用白色画出第一条 Bézier 曲线，接着，AutoRedraws 就会擦去原来的曲线。AutoRedraw 属性显著地降低了曲线的移动。你也许想通过其他的途径来对此作出改进。但是千万要小心！我试验过 AutoRedraw，第一条曲线总是会遗留下一些残点，即使坐标完全正确，也是如此。

BezDraw：开始 deCasteljau 算法。按 Inverse 模式绘制此段及后面的曲线。当触发了 MouseMove 事件时，能保证曲线是黑色的。

3.4 重画 Bézier 控制线

控制线是一条点线，该线连接图柄和曲线的端点。当使用两个图柄中的任一个来改变曲线的形状时，控制线也必须重画（见清单 3.4）。

清单 3.4 移动图柄 1 时重画 Bézier 曲线

```
Sub BezHandle1_MouseMove (Button As Integer, Shift As Integer, X!, Y!)
OFBN = 45 + NodeZoom
Xstr = Format $ (X \ PixelX, "000")
Ystr = Format $ (Y \ PixelY, "000")
XYlabel.Caption = "[" + Xstr - " , " + Ystr - "]"
If Button Then
    BezDraw
    ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
    ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
    ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
    h1.X = X
    h1.Y = Y
    Shape1.Left = h1.X - OFBN
    Shape1.Top = h1.Y - OFBN
    Shape1.Visible = True
    ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
    ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
    BezDraw
End If
End Sub
```

OFBN：Bézier 节点的偏移（45 个 twip 正好是 shape1 宽度的一半）。移动过程中，此值将 shape1 置于 X，Y（鼠标指针位置）处

NodeZoom：当 ArtWork 窗口视图的比例因子由于放大（zoom in）或缩小（zoom out）而发生变化时，该值用来调整 OFBN 位移值，此值由 ZoomBar 控制对象给定。

Xstr 和 Ystr：BezHandle 是一个独立的图片框，因此将 X，Y 坐标传到 XYlabel，以显示正确的坐标值。

If Button then：如果鼠标按钮被按住，则继续。

DrawBez：以 Inverse(6)模式画一段曲线。

ArtWork. DrawMode：将 DrawMode 改为 Xor(7)，这样控制线(点线)经过其他对象时就不会将它们擦掉。

ArtWork. Line：画点线。在 Xor 模式下，黄色看上去像是青色。有两个不同的图柄，所以必须画两条单独的控制线。

h1. X=X：交换当前 X 点，制造一种移动的错觉。

h1. Y=Y：交换当前 Y 点，制造一种移动的错觉。

Shape1. Left，Shape1. Top：用 Shape 控制对象取代 BezHandle1 图片框控制对象。如果移动 BezHandler 控制对象，则它会不停地弹回当前 X 坐标处。另外再写一些代码可以避免这种现象的出现，但是为什么要如此麻烦呢？Shape 控制对象就很好，它还省去编写用以画出小图柄或节点形状的 API Polyline 或 Basic Line 语句。

Shape1. Visible=True：因为本程序只用了两个 shape 控制对象，它们的位置可能是未知的。将它们移到恰当的位置，并使它们可见。

BezDraw：在 Inverse(6)模式下绘制一段曲线。这将擦去上一幅曲线图象，制造一种移动的错觉。

ScaleLeft 属性：为某个窗体、图片框或打印机对象设置新的 X 坐标。给定的 X 位置(水平方向)从控制对象的最左边开始(ScaleLeft 是控制对象内部最左端的坐标值)。

ScaleTop 属性：为某个窗体、图片框或打印机对象设置新的 Y 坐标。给定的 Y 位置(垂直方向)从控制对象的最上边开始(ScaleTop 是控制对象内部的最顶部的坐标值)。

用 ScaleMode 可改变这些设置。更详细的内容请参见 Visual Basic 语言参考手册。

3.5　存储曲线的新坐标点

当一条曲线被拉伸或移动到新的位置后，坐标点必须保存起来。所有节点和图柄都必须重新定位，控制线也必须重画(见清单 3.5)。

清单 3.5　重新设定比例将 Bézier 图柄移至新位置

```
Sub BezHandle1_MouseUp (Button As Integer, Shift As Integer, X!, Y!)
OFBN = 45 + NodeZoom
Shape1.Visible = False
BezHandle1.Move X - OFBN, Y - OFBN
BezHandle1.ScaleLeft = BezHandle1.Left
BezHandle1.ScaleTop = BezHandle1.Top
BezHandle1.Visible = True
ArtWork.DrawStyle = 0: ArtWork.DrawWidth = 1
    p1X(B) = p1.X: p1Y(B) = p1.Y
    p2X(B) = p2.X: p2Y(B) = p2.Y
    h1X(B) = h1.X: h1Y(B) = h1.Y
    h2X(B) = h2.X: h2Y(B) = h2.Y
    BezierStore
    UpdateObjects
```

```
p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
End Sub
```

OFBN:Bézier 节点的偏移(45 twip 是 shape1 宽度的一半)。移动过程中,该值将 shape1 置于 X,Y(鼠标指针位置)处。

Shape1. Visible:移动后,隐藏 Shape 控制对象。

BezHandle1. Move:将图柄移至新位置。

BezHandle1. ScaleLeft:参见 BezHandle. ScaleTop。

BezHandle1. ScaleTop:重新设定图片框的比例,以使得 X,Y 坐标与 ArtWork 相同。因为 BezHandle 图片框位于另一个图片框内,所以,它的 Left,Top 属性都是用 ArtWork 的内部比例来定位的。将 BezHandle1 的 ScaleLeft 和 ScaleTop 设成等于它自己的左边和顶部位置,就会使得 BezHandle1 的内部 X,Y 坐标等于 ArtWork 的比例。

BezHandle1. Visible:该图柄在 MouseDown 事件中被隐藏起来了,但没有移动,现在必须把它移到曲线所在的新位置上。

ArtWork. DrawStyle:现在使用的是点线模式,因此该句将其改为实线模式。

ArtWork. DrawWidth:确保画图宽度切换回了默认值 1。

p1X(B)和 p1Y(B),等等……:存储图柄和节点。这些数组变量的详细情况在清单 2.8 中解释。

BézierStore:存储边界方框的新坐标位置。

p1X,p1Y……:重设 BezHandle 和 BezNode 位置。

UpdateObjects:重画到目前为止所创建的所有对象。

ArtWork. DrawMode, ArtWork. DrawStyle:又准备画点线,所以将 DrawMode 设成 Xor,将 DrawStyle 改为 Dot(点线)。

ArtWork. Line:画出与曲线连接的每条控制线(点线)。

3.6 曲线的第二个 Bézier 图柄

BezHandle2 的所有鼠标事件都使用与 BezHandle1 鼠标事件相同的公式和过程。现在处理的是第二 Bézier 图柄和节点,所以在恰当的地方把这些名字改过来(见清单 3.6)。

清单 3.6　单击第二个图柄,设置 Bézier 曲线

```
Sub BezHandle2_MouseDown (Button As Integer, Shift As Integer, X!, Y!)
XX2 = E(B): YY2 = S(B): XX1 = W(B): YY1 = N(B)
    BezHandle2.Visible = False
BezNode2.BackColor = Blue: BezNode1.BackColor = Black
    BezMove
    BezDraw
End Sub
```

BezHandle2：有关 BezHandle2_MouseDown 事件的详细情况请参考清单 3.3 中的 BezHandle1_MouseDown 过程。

关于移动第二个图柄后如何重画 Bézier 曲线的细节请参见清单 3.7。

清单 3.7 移动图柄后重画 Bézier 曲线

```
Sub BezHandle2_MouseMove (Button As Integer, Shift As Integer, X!, Y!)
OFBN = 45 + NodeZoom

Xstr = Format$(X \ PixelX, "000")
Ystr = Format$(Y \ PixelY, "000")
XYlabel.Caption = "[" + Xstr + "," + Ystr + "]"

If Button Then
    BezDraw
    ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
    ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
    ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
        h2.X = X
        h2.Y = Y
    Shape2.Left = h2.X - OFBN
    Shape2.Top = h2.Y - OFBN
    Shape2.Visible = True
    ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
    ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
    BezDraw
End If
End Sub
```

BezHandle2：有关 BezHandle2_MouseMove 事件的细节请参考清单 3.4 中的 BezHandle1_MouseMove 过程。

有关重新缩放 Bézier 图柄的情况请见清单 3.8。

清单 3.8 重新设定比例，将 Bézier 图柄移至新位置

```
Sub BezHandle2_MouseUp (Button As Integer, Shift As Integer, X!, Y!)
OFBN = 45 + NodeZoom
Shape2.Visible = False
BezHandle2.Move X - OFBN, Y - OFBN
BezHandle2.ScaleLeft = BezHandle2.Left
BezHandle2.ScaleTop = BezHandle2.Top
BezHandle2.Visible = True
ArtWork.DrawStyle = 0: ArtWork.DrawWidth = 1
p1X(B) = p1.X: p1Y(B) = p1.Y
p2X(B) = p2.X: p2Y(B) = p2.Y
h1X(B) = h1.X: h1Y(B) = h1.Y
h2X(B) = h2.X: h2Y(B) = h2.Y
BézierStore
UpdateObjects
p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
```

```
ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
End Sub
```

p1X(B),p1Y(B)到 h2X(B),h2Y(B)存储 Bézier 曲线的 4 个屏幕控制位置(也就是 p1,p2,h1,h2)。

```
p1X(B)=p1.X;p1Y(B)=p1Y;p2X(B)=p2.X;p2Y(B)=p2.Y
h1X(B)=h1.X;h1Y(B)=p1Y;h2X(B)=h2.X;h2Y(B)=h2.Y
```

方框、直线和圆对象的屏幕位置可以很方便地用它们边界方框的坐标来存储,因为它们的边框(轮廓)正好达到它们边界方框的边框(轮廓)。另外 Bézier 曲线的控制点也位于一个边界方框区域内,因此在画出 Bézier 曲线后,必须存储四个控制点的 X,Y 坐标(P1,P2,H1,H2)。附带磁盘的 16MISC 目录下的 Bounding 程序展示了在一个随机生成的边界方框内有无数个可能的曲线控制点坐标,它有助于你理解上面的概念。

BézierStore:存储了上述的点后,就必须调用 BézierStore 过程计算出该曲线边界方框的位置。该过程能够算出屏幕上的四个曲线控制点哪一个是在最北边,哪一个在最西边,哪一个在最南边,哪一个在最东边,从而计算出一个正好包容该曲线对象的边界方框。

3.7 将一条直线转换成一条曲线

BezHandle 过程给现有的直线加上 Bézier 节点和图柄,将它转换成一条曲线。

使用工具按钮中的 line(直线)工具(按钮 3)画一条直线。用指针工具(按钮 1)选定直线或在直线周围画一个选定方框以选定该直线,然后,选定 Bézier 工具(按钮 2)并双击该直线。

一个小窗体就会显示出来,你可以从中选择一个按钮将该直线转换成曲线。

接着,ArtWork.Dbl_Click 事件调用 BezHandle 过程在直线的两端各放置一个 Bézier 节点,最后,原来的直线的对象编号就从 FlagLine 切换到 FlagBézier(见清单 3.9)。

清单 3.9 在一条直线上设置图柄和点线

```
Sub BezHandles ()
OFBN = 45 + NodeZoom
    BezNode1.Move W(B) - OFBN, N(B) - OFBN
    BezNode2.Move E(B) - OFBN, S(B) - OFBN
p1.X = W(B)
p1.Y = N(B)
p2.X = E(B)
p2.Y = S(B)
''''''''''''''
ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
HalfLeft = (E(B) + W(B)) \ 2
HalfTop = (S(B) + N(B)) \ 2
     BezHandle1.Left = (W(B) + HalfLeft) \ 2 - OFBN
     BezHandle1.Top = (N(B) + HalfTop) \ 2 - OFBN
     BezHandle2.Left = (E(B) + HalfLeft) \ 2 - OFBN
```

```
        BezHandle2.Top = (S(B) + HalfTop) \ 2 - OFBN
    BezHandle1.ScaleLeft = BezHandle1.Left
    BezHandle1.ScaleTop = BezHandle1.Top
    BezHandle2.ScaleLeft = BezHandle2.Left
    BezHandle2.ScaleTop = BezHandle2.Top
    BezNode1.ScaleLeft = BezNode1.Left
    BezNode1.ScaleTop = BezNode1.Top
    BezNode2.ScaleLeft = BezNode2.Left
    BezNode2.ScaleTop = BezNode2.Top
    ReScale 'If a Zoom Tool is to be used in program
    h1.X = BezHandle1.Left + OFBN
    h1.Y = BezHandle1.Top + OFBN
    h2.X = BezHandle2.Left + OFBN
    h2.Y = BezHandle2.Top + OFBN
    p1X(B) = p1.X: p1Y(B) = p1.Y
    p2X(B) = p2.X: p2Y(B) = p2.Y
    h1X(B) = h1.X: h1Y(B) = h1.Y
    h2X(B) = h2.X: h2Y(B) = h2.Y
    ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
    ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
    End Sub
```

OFBN：Bézier Handle1 和 Bézier Handle2 的偏移(45 twip 是图柄宽度的一半)。

ArtWork. AutoRedraw：因为不需要捕获该过程中的任何图象，所以将该属性关闭。

BezNode1. Move，BezNode2. Move：用 Bézier 节点替换两个标准直线节点。

p1. X，p1. Y，p2. X，p2. Y：这些值代表曲线的起点和终点，所以将它们设成与直线的起点和终点的值(这些值先前存在 W(B)、N(B)、S(B)和 E(B)中)相等。

ArtWork. DrawMode：加入连接 BezHandle 和 BezNode 的点线，这些线按 Xor 模式绘制。同时将 DrawStyle 改为 Dot(点线)。

HalfLeft，HalfTop：必须算出直线的中间点，所以将图柄置于靠近直线中心的位置。

BezHandle1. Left，BezHandle1. Top：参见 BezHandle2. Left 和 BezHandle2. Top。

BezHandle2. Left，BezHandle2. Top：图柄就放在该位置。

BezHandle1. ScaleLeft(Top)，BezHandle2. ScaleLeft(Top)：参见 BezNode1. ScaleLeft(Top)，BezNode2. ScaleLeft(Top)。

BezNode1. ScaleLeft(Top)，BezNode2. ScaleLeft(Top)：重新缩放 Bézier 曲线的两个图柄及两个节点，以反映 ArtWork 的内部 X、Y 坐标比例。

Rescale：如果使用了 zoom(缩放)工具，就需要反映出先前放大或缩小前桌面的比例。

h1. X，h1. Y，h2. X，h2. Y：图柄现在已正确定位，因此将它们的 X、Y 坐标传给存放这两个图柄当前位置的变量。

p1X(B)，p2X(B)，h1X(B)，h2X(B)等等：现在将所有四个 Bézier 控制点存入数组中，记录哪些点属于哪条 Bézier 曲线。

ArtWork. Line：最后，画出连接两个 Bézier 节点和两个 Bézier 图柄的控制线(点线)。

当用 Bézier 工具(按钮 3)双击一条直线后，屏幕的中央就会显示出 Node Edit 窗体。如果当前选定的对象是一条直线，那么请选择 To Curve 按钮。如果当前选定的对象是一条曲

线，那么请选择 To Line 选项按钮。图 3.9 显示了当用户选择转换成曲线后，直线看上去是什么样子。

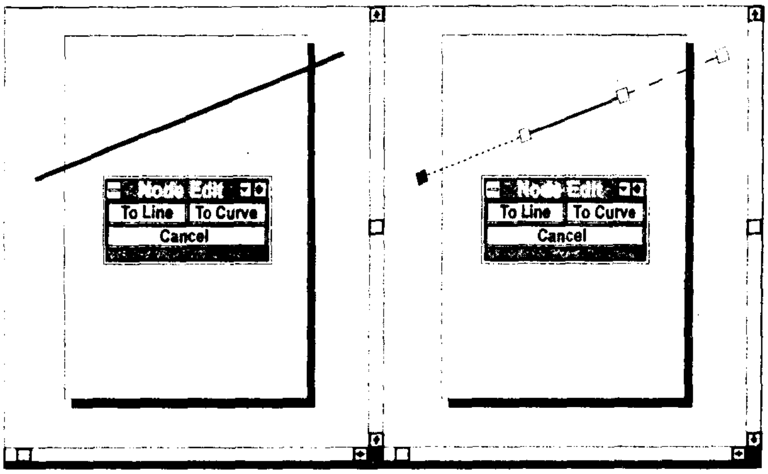


图 3.9　一条转换成曲线的直线

3.8　捡取一条曲线

位于一条曲线对象两端的节点叫做 BezNode1 和 BezNode2，可以用这两个节点中的任意一个来移动曲线。

BezNode1 的鼠标器事件过程是以 BezHandle 过程（从清单 3.3 开始）为基础的，移动曲线时，还必须移动与 BezNode1 相连的图柄（BezHandle1）（见清单 3.10）。

清单 3.10　单击 Bézier 节点 1 时设定曲线

```
Sub BezNode1_MouseDown (Button As Integer, Shift As Integer, X!, Y!)
XX2 = E(B): YY2 = S(B): XX1 = W(B): YY1 = N(B)

BezNode1.BackColor = Blue: BezNode2.BackColor = Black
    X1 = p1.X: Y1 = p1.Y
    XX1 = X: YY1 = Y
    P = 1
End Sub
```

X1，Y1：在移动之前临时存放当前节点位置。

XX1，YY1：决定是真正移动节点还是仅仅将其高亮显示。

BezNode1 MouseMove 事件是以 BezHandle1 MouseMove 事件过程为基础的。对清单 3.11 中代码更进一步的解释请参见清单 3.4。

清单 3.11　移动 Bézier 节点 1 后重画 Bézier 曲线

```
Sub BezNode1_MouseMove (Button As Integer, Shift As Integer, X!, Y!)
OFBN = 45 + NodeZoom
Xstr = Format $ (X \ PixelX, "000")
```

```
Ystr = Format $ (Y \ PixelY, "000")
XYlabel.Caption = "[" + Xstr + " , " + Ystr + "]"
If Button Then
    If X = XX1 And Y = YY1 Then XX1 = 0 And YY1 = 0: Exit Sub
      If P = 1 Then
        P = 0: BezHandle1.Visible = True: BezHandle2.Visible = True: BezMove
        BezDraw
    End If
BezDraw
ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
    p1.X = X
    p1.Y = Y
    h1.X = (X - X1) + h1X(B)
    h1.Y = (Y - Y1) + h1Y(B)
BezHandle1.Left = h1.X - OFBN
BezHandle1.Top = h1.Y - OFBN
    Shape1.Left = p1.X - OFBN
    Shape1.Top = p1.Y - OFBN
    BezNode1.Visible = False: Shape1.Visible = True
    ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
    ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
BezDraw
End If
End Sub
```

X=XX1 和 Y=YY1：如果单击了一个节点却并没有移动它，那么相等为真。这可以防止 Bézier 节点移动若干象素位置。

If P=1 Then：对于每条新近转换成的曲线，P 为真。必须调用 BezMove 过程（仅一次）以擦除原来的直线并开始画 Bézier 曲线。

将 P 设成等于 0（P=0）可以使该过程跳过它后面的语句块，因为该语句块只需执行一次：

```
h1.X=(X-X1)+h1X(B);h1.Y=(Y-1)+h1Y(B);
```

上面的计算使图柄移动与节点移动的距离相同。Y 等于鼠标移动时的当前 Y 位置，Y1 是移动的起始位置（节点位置）。将这两个值相减，再加上每次移动前的图柄位置，就可以使图柄和节点之间的距离保持不变。

BezNode1_MouseUp 事件是以 BezHandle1_Mouse_Up 过程为基础的。对清单 3.12 中代码的更进一步的解释请参见清单 3.5。

清单 3.12 重新设定比例，将 Bézier 图柄移至新位置

```
Sub BezNode1_MouseUp (Button As Integer, Shift As Integer, X!, Y!)
OFBN = 45 + NodeZoom
If X = XX1 And Y = YY1 Then XX1 = 0 And YY1 = 0: Exit Sub
BezNode1.Move X - OFBN, Y - OFBN
BezNode1.ScaleLeft = BezNode1.Left
```

```
BezNode1. ScaleTop = BezNode1. Top
Shape1. Visible = False: BezNode1. Visible = True
BezHandle1. ScaleLeft = BezHandle1. Left
BezHandle1. ScaleTop = BezHandle1. Top
X1 = p1. X: Y1 = p1. Y
    p1X(B) = p1. X: p1Y(B) = p1. Y
    p2X(B) = p2. X: p2Y(B) = p2. Y
    h1X(B) = h1. X: h1Y(B) = h1. Y
    h2X(B) = h2. X: h2Y(B) = h2. Y
UpdateObjects
p1. X = p1X(B): p1. Y = p1Y(B): p2. X = p2X(B): p2. Y = p2Y(B)
h1. X = h1X(B): h1. Y = h1Y(B): h2. X = h2X(B): h2. Y = h2Y(B)
ArtWork. DrawMode = 7: ArtWork. DrawStyle = 2
    ArtWork. Line (p1. X, p1. Y)-(h1. X, h1. Y), YELLOW
    ArtWork. Line (p2. X, p2. Y)-(h2. X, h2. Y), YELLOW
BézierStore
End Sub
```

本章中没有列出 BezNode2 的鼠标器事件，请参考附带磁盘上 Artisan 程序的源代码。

BezNode2 的鼠标器事件过程是以 BezHandle 过程（从清单 3.3 开始）为基础的。移动曲线时还必须移动与 BezNode2 相连接的图柄 BezHandle2。

BezNode2_MouseDown，BezNode2_MouseMove 和 BezNode2_MouseUp 事件所用的公式和过程与 BezNode1 的鼠标事件相同，唯一的差别是 P1 和 H1 的值改为 P2 和 H2 的值。

3.9　移动前擦除曲线图象

清单 3.13 中的过程在某个图柄或节点移动后被调用来画出第一条 Bézier 曲线。当一条曲线的位置确定了之后，就用 UpdateObjects 过程来画出它，UpdateObjects 过程是按 Copy_Pen(13)模式画图的。使用 AutoRedraw 功能，曲线的图象就会保留下来。当编辑或移动曲线后，就应该把保留的曲线图象从屏幕上擦掉。

清单 3.13　擦除旧的曲线图象和图柄线(点线)

```
Sub BezMove ()
ArtWork. DrawMode = 7: ArtWork. DrawStyle = 2
ArtWork. Line (p1. X, p1. Y)-(h1. X, h1. Y), YELLOW
ArtWork. Line (p2. X, p2. Y)-(h2. X, h2. Y), YELLOW
If BezErase = True Then
    ArtWork. AutoRedraw = True
    ArtWork. DrawStyle = 0: ArtWork. DrawMode = 13
    ArtWork. Line (p1. X, p1. Y)-(p2. X, p2. Y), QBColor(15)
    ArtWork. AutoRedraw = False: BezErase = False
Else
    ArtWork. AutoRedraw = True
    ArtWork. DrawMode = 16
    ArtWork. DrawWidth = 1
    ArtWork. PSet (0, 0)
    BezDraw
    ArtWork. DrawWidth = 1
    ArtWork. AutoRedraw = False
End If
```

```
ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
End Sub
```

ArtWork. DrawMode：连接节点和图柄的控制线（点线）是按 Xor 模式绘制的。

If BezErase=True Then：当第一次将直线转换成曲线时，该标记被设置，需要画一条白色的直线以擦去原来的直线。

Else：该曲线不像一条直线，因此，将 DrawMode 改为 White(16)，以准备画一条白色的曲线。

ArtWork. Pset(0,0)：当 AutoRedraw 为 True 时，Visual Basic 的 DrawMode 在不同绘图模式之间的切换是自动进行的，并没有给出提示。通过在窗体上执行任何一条绘图语句，即可重新恢复原来的绘图模式。此处的 Pset 语句就是这样的例子，该语句将当前象素的坐标设置为(0,0)，离开显示区域。这种方法对在不同绘图模式之间经常切换的 Windows API 图形函数仍然适用。

BezDraw：绘制 Bézier 曲线

ArtWork. Line：最后，重新画出先前擦掉的点线。

3.10 为曲线建立一个边界方框

BézierStore 过程检查曲线节点及图柄的所有可能位置，然后仅存储能够包住该曲线的最远的 4 个坐标。

可以用一个数组排序函数来完成此任务，而且代码较少。如果你有 Visual Basic 3.0 或更高的版本，就可以用 Choose 语句来缩短该过程。也可以用嵌套循环来压缩该过程代码。BézierStore 按四个部分给出了此过程，这样你就会知道需要做的是什么（见清单 3.14）

清单 3.14 计算曲线的新边界方框

```
Sub BézierStore ()
If p1.X <= h1.X And p1.X <= h2.X And p1.X <= p2.X Then W(B) = p1.X
If h1.X <= p1.X And h1.X <= h2.X And h1.X <= p2.X Then W(B) = h1.X
If h2.X <= p1.X And h2.X <= h1.X And h2.X <= p2.X Then W(B) = h2.X
If p2.X <= p1.X And p2.X <= h1.X And p2.X <= h2.X Then W(B) = p2.X
If p1.Y <= h1.Y And p1.Y <= h2.Y And p1.Y <= p2.Y Then N(B) = p1.Y
If h1.Y <= p1.Y And h1.Y <= h2.Y And h1.Y <= p2.Y Then N(B) = h1.Y
If h2.Y <= p1.Y And h2.Y <= h1.Y And h2.Y <= p2.Y Then N(B) = h2.Y
If p2.Y <= p1.Y And p2.Y <= h1.Y And p2.Y <= h2.Y Then N(B) = p2.Y
If p1.X >= h1.X And p1.X >= h2.X And p1.X >= p2.X Then E(B) = p1.X
If h1.X >= p1.X And h1.X >= h2.X And h1.X >= p2.X Then E(B) = h1.X
If h2.X >= p1.X And h2.X >= h1.X And h2.X >= p2.X Then E(B) = h2.X
If p2.X >= p1.X And p2.X >= h1.X And p2.X >= h2.X Then E(B) = p2.X
If p1.Y >= h1.Y And p1.Y >= h2.Y And p1.Y >= p2.Y Then S(B) = p1.Y
If h1.Y >= p1.Y And h1.Y >= h2.Y And h1.Y >= p2.Y Then S(B) = h1.Y
If h2.Y >= p1.Y And h2.Y >= h1.Y And h2.Y >= p2.Y Then S(B) = h2.Y
If p2.Y >= p1.Y And p2.Y >= h1.Y And p2.Y >= h2.Y Then S(B) = p2.Y
End Sub
```

BézierStore：此过程搜索 Bézier 曲线四个控制点的数据数组，其格式如下：

p1.X,p1.Y(BezNode1)
h1.X,h1.Y(BezHandle1)
h2.X,h2.Y(BezHandle2)
p2.X,p2.Y(BezNode2)

当 BézierStore 找到最西边，最北边，最东边和最南边点后，它就将这些值分别存储在 W(B)，N(B)，E(B)和 S(B)中，最后这些值用来确定对象的边界，也就是它的边界方框。

图 3.10 显示了 6 条曲线，每条曲线的 BezNodes 和 BezHandles 坐标都不同。因为放置 4 个曲线控制点的方法是无穷的，所以必须找到一种方法在生成一条曲线后重新定位其边界方框。

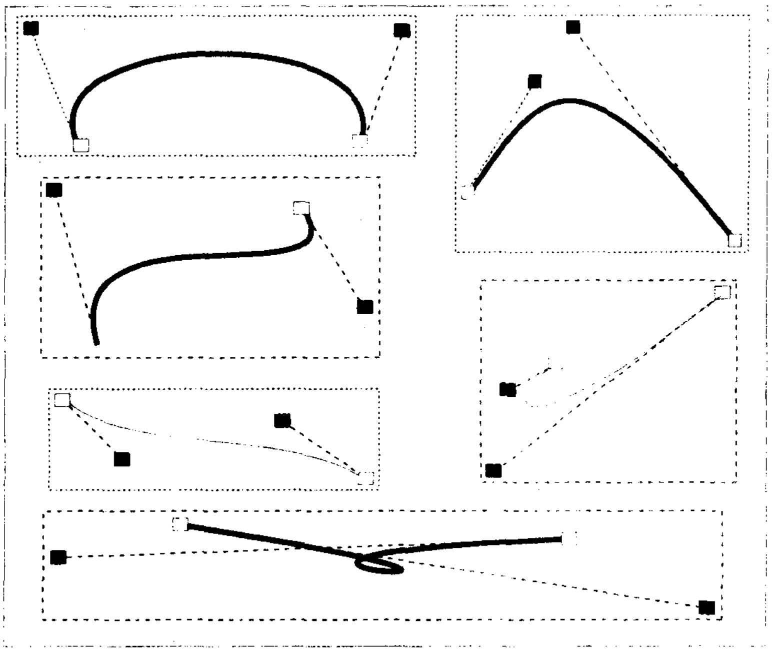


图 3.10 W,N,E,S 值的几种方案

3.11 曲线上控制线(lever lines)的视觉状态

VState 变量是很重要的，它有时让人很迷惑。VState 将 Bézier 节点和图柄的 Visible(可见)属性打开或关闭，当因为下面的任何一种操作而使得 Bézier 曲线失去了聚焦时，这对于视觉外观来说是非常必要的：

- 选定了别的工具
- 选择了一种颜色
- 滚动 ArtWork
- 在曲线的边界方框外单击桌面

有关重新设置比例及曲线图柄线的情况请参见清单 3.15。

清单 3.15　重新设置比例和曲线的图柄线（点线）

```
Sub BezUpdate ()
OFBN = 45 + NodeZoom
p1.X = p1X(B): p1.Y = p1Y(B)
p2.X = p2X(B): p2.Y = p2Y(B)
h1.X = h1X(B): h1.Y = h1Y(B)
h2.X = h2X(B): h2.Y = h2Y(B)
BezHandle1.Left = h1.X - OFBN: BezHandle1.Top = h1.Y - OFBN
BezHandle2.Left = h2.X - OFBN: BezHandle2.Top = h2.Y - OFBN
BezNode1.Left = p1.X - OFBN: BezNode1.Top = p1.Y - OFBN
BezNode2.Left = p2.X - OFBN: BezNode2.Top = p2.Y - OFBN
BezHandle1.ScaleLeft = BezHandle1.Left
BezHandle1.ScaleTop = BezHandle1.Top
BezHandle2.ScaleLeft = BezHandle2.Left
BezHandle2.ScaleTop = BezHandle2.Top
BezNode1.ScaleLeft = BezNode1.Left
BezNode1.ScaleTop = BezNode1.Top
BezNode2.ScaleLeft = BezNode2.Left
BezNode2.ScaleTop = BezNode2.Top
Shape1.Left = h1.X - OFBN: Shape1.Top = h1.Y - OFBN
Shape2.Left = h2.X - OFBN: Shape2.Top = h2.Y - OFBN
ReScale ''' If ZoomTool Used
ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), YELLOW
ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), YELLOW
ArtWork.DrawStyle = 0
If ShowDotted = False Then VState = False Else VState = True
BezHandle1.Visible = VState: BezHandle2.Visible = VState
If Not CurrentTool = 1 Then VState = False Else VState = True
BezNode1.Visible = VState: BezNode2.Visible = VState
If CurrentTool = 1 Then N_H_Clear Else Nodes
End Sub
```

OFBN：BezHandle1 和 BezHandle2 以及 BezNode1 和 BezNode2 的偏移值。(45 twip 是每个控制对象的宽度的一半)。该值确定每个控制对象的中心点位置。

p1.X,p1.Y,p2.X,p2.Y：这些值表示曲线的起点和终点。索引数组 p1X(B),p1Y(B)等存储每条曲线所需要的值。

h1.X,h1.Y,h2.X,h2.Y：这些值代表两个 Bézier 图柄的值。索引数组 h1X(B),h1Y(B)等存储每条曲线所需的值。

图 3.11 给出的是 ShowDotted 为真且节点均可见的情况，图 3.12 给出的是 ShowDotted 为假而节点均可见的情况，图 3.13 给出的则是 ShowDotted 为假且没有节点的情况。

调用 BezUpdate 过程显示或隐藏 Bézier 图柄、节点及连接每个控制对象的点线。这很大程度上取决于什么时候曲线失去聚焦。(例如，用户已经选定了曲线，但接着又选定了别的工具)。本程序中只用了四个图片框来代表图柄和节点。当曲线被选定时，必须重新确定这些控制对象的位置。还必须重新设定每一个节点和图柄的内部尺寸比例，以与 ArtWork 图片框的当前比例同步。

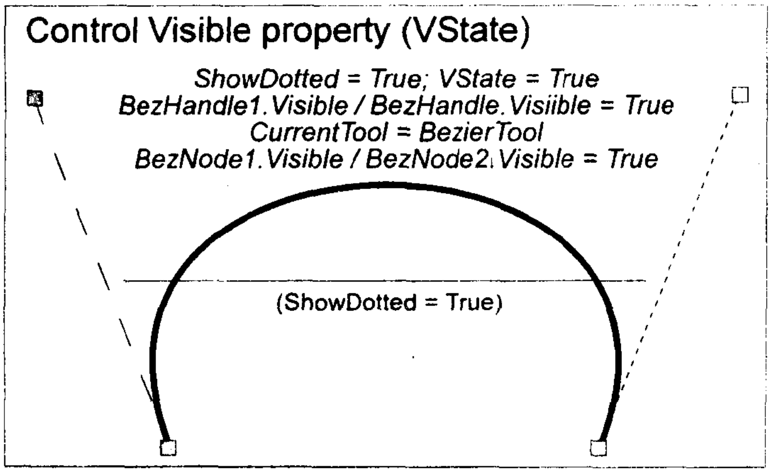

Control Visible property (VState)
ShowDotted = True; VState = True
BezHandle1.Visible / BezHandle.Visiible = True
CurrentTool = BezierTool
BezNode1.Visible / BezNode2.Visible = True
(ShowDotted = True)

图 3.11 ShowDotted 为真，节点可见

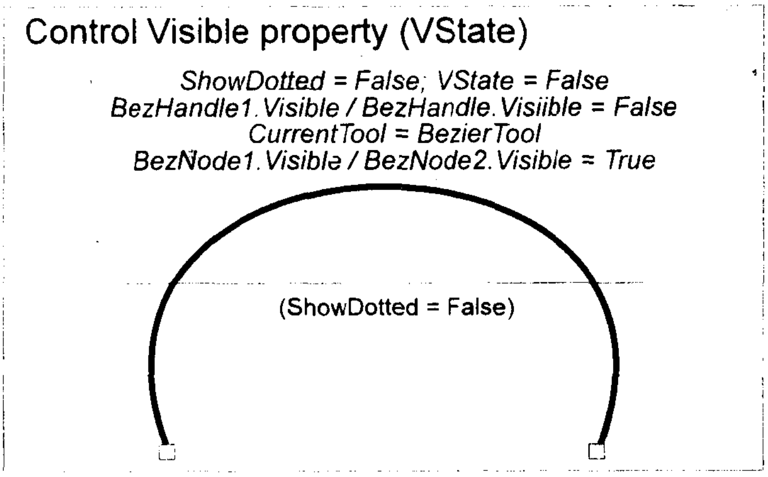

Control Visible property (VState)
ShowDotted = False; VState = False
BezHandle1.Visible / BezHandle.Visiible = False
CurrentTool = BezierTool
BezNode1.Visible / BezNode2.Visible = True
(ShowDotted = False)

图 3.12 ShowDotted 为假，节点可见

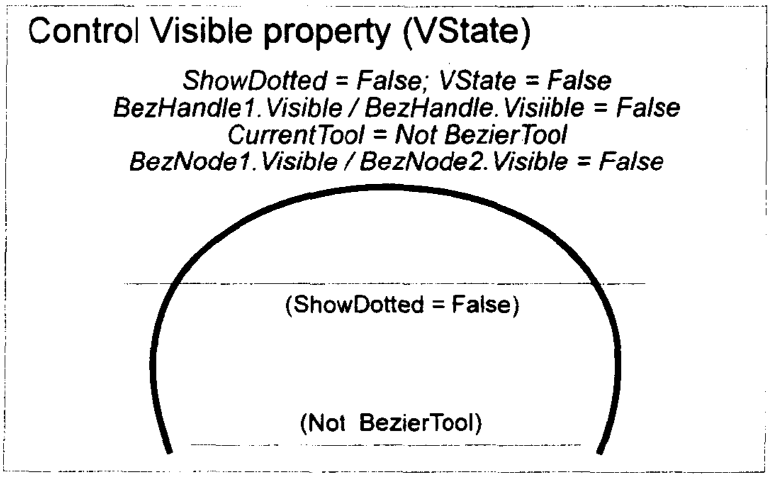

Control Visible property (VState)
ShowDotted = False; VState = False
BezHandle1.Visible / BezHandle.Visiible = False
CurrentTool = Not BezierTool
BezNode1.Visible / BezNode2.Visible = False
(ShowDotted = False)
(Not BezierTool)

图 3.13 ShowDotted 为假，没有节点

BezHandle1. Left(Top,ScaleLeft,ScaleTop)，等等：完成上面提到的任务。此时也可以移动两个 Shape 控制对象。

ArtWork. Line：画出连接每个图柄和节点的点线。该语句使得点线可见或不可见，这取决于它们在屏幕上的视觉状态。

附注：必须根据用户的上一次操作(失去焦点或取得焦点)来隐藏或者显示图柄和节点。

图 3.11、3.12，3.13 给出了一条曲线几种可能的视觉状态(VState)，这取决于标记 ShowDotted 的值。如果当前选定的不是 Bézier 工具，则 BezNodes 必须隐藏起来。

如果当前工具是 Bézier 工具，清除标准节点和拉伸图柄；如果当前选定的工具不是 Bézier 工具，则只显示标准节点。

标准节点是白色的节点，它们仅在方框和直线形状中出现。拉伸图柄是出现在对象的边界方框周围的 8 个红色小方框。拉伸图柄用来扩展或收缩所画的图象。

3.12　拉伸 Bézier 曲线

注意：如果任何图柄或节点的 X，Y 坐标与别的点的 X，Y 坐标相等，那么“被 0 除”就是一个很普遍的错误。

Twip 模式大大地减少了这种可能性，但是，让这种错误产生本身就是编程能力不强的体现。

当试图用一个值(分子)除以 0(分母)时就会发生“被 0 除”错误。假设在运行的时候，X 的值最后和 X1 的值相同，那么语句 354\(X－X1)就变成了 354\0。“被 0 除”错误会使程序终止。

BezStretch 过程没有努力防止“被 0 除”错误的产生，因为它采用的是基本的数学方法。不过，在本过程中是很难产生“被 0 除”错误的。

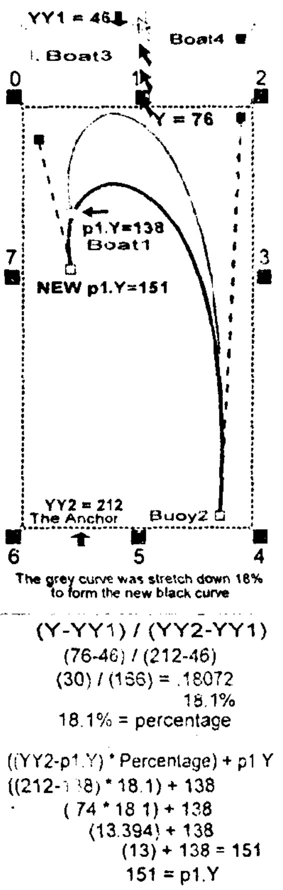

YY1 = 46
Boat4
Boat3
0
1
2
Y = 76
p1.Y=138
Boat1
7
3
NEW p1.Y=151
YY2 = 212
The Anchor
Buoy2
6
5
4
The grey curve was stretch down 18% to form the new black curve
(Y-YY1) / (YY2-YY1)
(76-46) / (212-46)
(30) / (166) = .18072
18.1%
18.1% = percentage
((YY2-p1.Y) * Percentage) + p1 Y
((212-138) * 18.1) + 138
(74 * 18 1) + 138
(13.394) + 138
(13) + 138 = 151
151 = p1.Y

图 3.14　拉伸或翻转一条 Bézier 曲线

图 3.14 给出了拉伸或翻转 Bézier 曲线所用的变量和公式。

清单 3.16 中的 BezStretch 过程收缩、扩展或翻转桌面上的曲线。拉伸仅在被扩展或收缩的边界方框的边上进行。边界方框的边在拉伸过程保持不动，它们的作用像抛锚点，而边界方框内的各个点(节点)就像是漂离抛锚点的船。船(节点)甚至从抛锚点漂离得更远。这种效果给人一种在拉伸图象时带有弹性的感觉。

清单 3.16　计算拉伸后曲线的新坐标

```
Sub BezStretch (Index As Integer， Button As Integer， Shift As Integer， X!， Y!)
Select Case Index
    Case 0， 1， 2， 4， 5， 6
      pt1 = p1Y(B)： pt2 = p2Y(B)： hd1 = h1Y(B)： hd2 = h2Y(B)
    Case 3， 7
      pt1 = p1X(B)： pt2 = p2X(B)： hd1 = h1X(B)： hd2 = h2X(B)
End Select
If pt1 = pt2 And hd1 = hd2 Then
W(B) = XX1： N(B) = YY1： E(B) = XX2： S(B) = YY2： Exit Sub
End If
Boat1 = pt1： Boat2 = pt2： Boat3 = hd1： Boat4 = hd2
''''''''''''''''''''''''''''''''''''''''''''
Select Case Index
    Case 0， 1， 2
       percentage! = (Y - YY1) / (YY2 - YY1)： ANCHOR = YY2
    Case 3
       percentage! = (X - XX2) / (XX1 - XX2)： ANCHOR = XX1
    Case 4， 5， 6
       percentage! = (Y - YY2) / (YY1 - YY2)： ANCHOR = YY1
    Case 7
       percentage! = (X - XX1) / (XX2 - XX1)： ANCHOR = XX2
End Select
Boat1 = ((ANCHOR - Boat1) * percentage!) + Boat1
Boat2 = ((ANCHOR - Boat2) * percentage!) + Boat2
Boat3 = ((ANCHOR - Boat3) * percentage!) + Boat3
Boat4 = ((ANCHOR - Boat4) * percentage!) + Boat4
If Index = 3 Or Index = 7 Then GoTo Xpoint Else GoTo Ypoint
Xpoint： 'Stretch from middle side Handles
p1.X = Boat1： p1X(B) = Boat1： p2.X = Boat2： p2X(B) = Boat2
h1.X = Boat3： h1X(B) = Boat3： h2.X = Boat4： h2X(B) = Boat4
BézierStore
Exit Sub
Ypoint： 'Stretch from middle top and bottom Handles
p1.Y = Boat1： p1Y(B) = Boat1： p2.Y = Boat2： p2Y(B) = Boat2
h1.Y = Boat3： h1Y(B) = Boat3： h2.Y = Boat4： h2Y(B) = Boat4
If Index = 1 Or Index = 5 Then BézierStore： Exit Sub
''''''''''''' Stretch from corner Handles '''''''''''''
pt1 = p1X(B)： pt2 = p2X(B)： hd1 = h1X(B)： hd2 = h2X(B)
Boat1 = pt1： Boat2 = pt2： Boat3 = hd1： Boat4 = hd2
Select Case Index
    Case 0， 6
       percentage! = (X - XX1) / (XX2 - XX1)： ANCHOR = XX2
    Case 2， 4
       percentage! = (X - XX2) / (XX1 - XX2)： ANCHOR = XX1
    Case Else
       BézierStore
       Exit Sub
End Select
Boat1 = ((ANCHOR - Boat1) * percentage!) + Boat1
Boat2 = ((ANCHOR - Boat2) * percentage!) + Boat2
Boat3 = ((ANCHOR - Boat3) * percentage!) + Boat3
```

```
Boat4 = ((ANCHOR - Boat4) * percentage!) + Boat4
p1.X = Boat1: p1X(B) = Boat1: p2.X = Boat2: p2X(B) = Boat2
h1.X = Boat3: h1X(B) = Boat3: h2.X = Boat4: h2X(B) = Boat4
BézierStore
End Sub
```

Select Case Index：在一条选定的曲线周围有 8 个图柄（红色方框），每个图柄有一个索引号，索引号从 0（左上角的红色方框）开始，顺时针方向编号至索引号 7。

Case0，1，2，4，5，6：如果用这些红方框中的某一个（顶行的 0，1，2 或底行的 4，5，6）来拉伸该曲线，临时变量 pt1，pt2，hd1，hd2 就用来存储曲线的两个图柄和节点的 Y 坐标。

Case 3，7：如果用曲线两边的两个红色方框（左边的方框在 0 和 6 之间，右边的在 2 和 6 之间）来拉伸该曲线，所做处理与前面一样，不同的是保存曲线的 X 坐标。

If pt1＝pt2 And hd1＝hd2 Then：曲线像是一条水平的或垂直的直线，不能拉伸，因此退出该子过程。

Boot1＝pt1：Boot2＝pt2：Boot3＝hd1：Boot4＝hd2：临时存放四个 Bézier 曲线控制点。这些点被叫做“船”，是因为当拉伸曲线的边界方框时，它就会漂浮到新的位置。

Select Case Index：根据使用的是哪一个红色方框，转至下面相应的 Case 表达式。

percentage!：计算边界方框移动的百分比。采用浮点（小数点）精度，所以该变量被标记为 single（!）类型。

Case 0，1，2：使用顶行中的某一个红色方框来拉伸边界方框。

Case 3：使用右边中间的那个红色方框来拉伸边界方框。

Case 4，5，6：使用底行中的某一个红色方框来拉伸边界方框。

Case 7：使用左边中间的那个红色方框来拉伸边界方框。

Boat1＝（（ANCHOR－Boat1）＊percentage!）＋Boat1：公式的简单形式是这样的：（（YY2－P1.Y）＊Percentage!）＋P1.Y。

YY2 是“抛锚点”（ANCHOR），因为它不移动（固定的边界方框位置）。P1.Y 是“船”（Boat1），因为在拉伸时，它要浮动到新的位置。Percentage! 使用边界方框的当前位置和 Y 鼠标移动位置通过公式（Y－YY1）/（YY2－YY1）来计算出边界方框扩大或缩小的百分比。P1.Y 是执行本次计算前 BOAT1 的初始位置。

If Index＝3 or Index＝7 Then：如果使用的是左边中间的方框（索引为 3）或右边中间的方框（索引为 7），则曲线只能在水平方向（X 轴）上拉伸，所以转至标号为 Xpoint 的行。

Else：否则，所使用的是除 3 和 7 之外的方框，则转至标号为 Ypoint 的行。

Xpoint：已经成功地在水平方向上拉伸了曲线，因此存储曲线的四个新坐标点。新位置用 BézierStore 过程来保存和存储。

Ypoint：已经成功地在垂直方向上拉伸了曲线，因此，存储曲线的四个新坐标点。新位置用 BézierStore 过程来保存和存储。只有使用的是垂直拉伸图柄才退出该过程。

如果使用角上的拉伸图柄，从水平方向和垂直方向同时拉伸该曲线，则用前面所用的相同的公式重新计算水平 X 轴，当然，所用的值应该是新值。

附注： 可以将 BézierStretch 过程中的大部分轴拉伸过程合并以缩短总代码量，清单

3.16 是以未经合并的方式给出的，这有助于理解算法。第 15 章中的 ArtAPI 程序在其 PolyStretch 过程中使用了本过程的一个翻版，以拉伸相互连接的多个图形。

3.13　本章小结

本章中，你接触了大量的计算和许多的变量名字。第一次接触 Bézier 曲线过程是难于掌握的。第 10 章给出了一个微型的 Bézier 曲线程序(Mini-Bézier)，它所用的变量名和过程都与本章相同。因为 Mini-Bézier 程序是一个独立的应用程序，所以不需要考虑可能出现在一个真正的绘图包中的所有其他的桌面功能。

在试图对 Artisan 程序或第 15 章中的 ArtAPI 程序作出修改之前，应该用在第 10 章中的 Mini - Bézier 程序练习一下。

如果想要缩短所有的 Bézier 曲线过程，那么可以试试递归的 Bézier 过程。在几乎所有的用 C 编写的绘图及文字转换程序中都采用了递归过程。

用 C 语言很容易写出递归的 Bézier 曲线过程，因为 C 可以指向数组并且在用 deCasteljau 算法计算每个曲线点时跟踪它们的位置。递归 Bézier 曲线名字的由来是由于它反复地调用 BezDivide(deCasteljau 公式)过程，将它自己的过程语句，BezDivide(P1,D1,E1,C1,leftCounter)和 BezDivide(C1,E1,D3,P2,rightCounter)放在 deCasteljau 公式之后。诀窍是在画出每个曲线点后从无限循环中跳出来。曲线上所画的点数取决于所执行的递归循环次数。

第 4 章　Artisan 程序——过程 C 到 N

本章按字母顺序从 C 至 N 解释 Artisan 程序中的过程。要讨论的过程中包括 ColorBar 调色板事件，这些事件可用来指定某个对象边框的颜色及填充颜色。在本章中可以找到其他的过程有以下这些：

FindAllObjects 过程与第 3 章“Bézier 曲线”中给出的 BézierStore 过程类似。当用指针工具围绕着一个对象画一个点线方框以选定该对象时，该过程扫描所有的边界方框坐标并选定（高亮显示）该区域内的第一个对象。

FindObject 过程与 FindAllObjects 过程非常相似，不同的是它仅选定（高亮显示）鼠标指针所在的边界方框。第 15 章“ArtAPI 程序”中，将采用一种更为复杂的方法，只有在接近（碰到）对象的边框形状时才高亮显示它。

GuideLines 过程很简单，它用一个特大的方框（Line 语句）造成一种正在拖动两个标尺上的页面标线的视觉效果。

Handle。Handles 和 HandleReverse 一起使用，用来重新确定对象周围拉伸方框的位置，拉伸方框是一些小红方框，它们被用来改变对象的大小。

直线及方框对象周围的节点（连接对象坐标点的小白框）定位方法可在 Nodes 过程中找到。最后，本章以 N_H_Clear 过程结束，该过程用以从桌面上清除或隐藏节点和图柄。

4.1 指定对象的边框和填充颜色

若要改变一个对象的边框颜色，则请用鼠标右键从 ColorBar 调色板中选定一种颜色。若要改变一个对象的填充颜色，则请用鼠标左键从 ColorBar 调色板中选定一种颜色。

边框颜色存储在变量 PenColor 中。如果用颜色填充某个对象，颜色值应存放在变量 BrushColor 中。ARTISAN7.BAS 文件中有一个全局过程 Update_Mouse_Colors，该过程可以获取正确的颜色，这些颜色存放在数组 Color 中。Update_Mouse_Color 过程从 Visual Basic 包含的 Iconworks 例子中修改得来。

Update_Mouse_Color 过程的代码清单位于 ARTISAN7.BAS 模块清单的尾部，该模块清单在附录 A 中。

清单 4.1　ColorBar_MouseDown 搜索选定的颜色值

```
Sub ColorBar_MouseDown (Button%, Shift%, X!, Y!)
    If(X>=0) And (X<=47) Then Update_Mouse_Colors Button,X,Y
End Sub
```

图片框 ColorBar 的 ScaleWidth 被设为 48。该图片框被 Display_Color_Palette 过程划分为若干个相等的部分，Display_Color_Palette 过程在 ARTISAN7.BAS 文件中，如下所示：

```
ColorBar.Scale(0,0)-(48,1)
```

```
For I=0 To 47
ColorBar.Line(I,0)-(I+1,1),Colors(I),BF
If I Then ColorBar.Line(I,0)-(I,3)
Next I
```

单击某种颜色就会调用 Update_Mouse_Color(ARTISAN7.BAS 文件)。鼠标的 X 位置舍入成一个整数，代表 Colors 数组的颜色索引号。Color 数组中的每种颜色都是十六进制的。

十六进制颜色值如：16777215，14737632 和 12632319。

上面的数字是 ARTISAN7.BAS 文件中通用声明段(标题为“Default Color Palette Values”)中定义的头 3 个颜色值的十六进制值。在 Form_Load 事件中，所有的数字被拼接起来生成 ColorBar 图片框中的 48 种颜色。

清单 4.2 ColorBar_Mouse 存储并显示选定的颜色

```
Sub ColorBar_MouseUp (Button%, Shift%, X!,Y!)
If Node(0).Visible = False Then Exit Sub
Select Case Button
    Case Right_Button
        OutLineColor(B) = PenColor
    Case Left_Button
        PaintColor(B) = BrushColor: FillValue(B) = 0
End Select
UpdateObjects
End Sub
```

If Node(0).Visible：如果节点可见，则退出鼠标器事件。

Case Right_Button：如果用鼠标右键选定了某种颜色，则将改变对象边框的颜色。

Case Left_Button：如果用鼠标左键选定了某种颜色，则将改变对象的填充颜色。

UpdateObjects：重画所有对象，以使颜色改变起作用。

清单 4.3 ColorBar_Paint 事件调用一个模块过程刷新所有的颜色

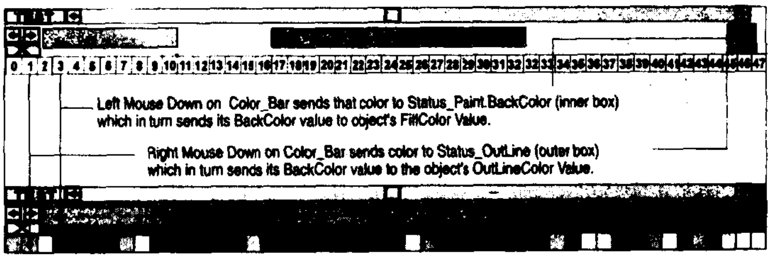
```
Sub ColorBar_Paint ()
    Display_Color_Palette ColorBar
End Sub
```

如果窗体被移动或被隐藏，Display_Color_Palette ColorBar 重新装入调色板内的全部颜色。参数 ColorBar 指明图片框控制对象，因为别的不同名字的控制对象也使用该过程。第 15 章中的 ArtAPI 程序将此调用从 Paint 事件中移了出去，这样就加快了颜色调色板的滚动速度。

清单 4.4 水平滚动条用来显示调色板中所有的颜色

```
Sub ColorScroll_Change ()
    ColorBar.Move ColorScroll.Value
End Sub
```

ColorScroll_Change 事件向左或向右移动 ColorBar，产生一种 ColorBar（颜色栏）固定不动而颜色在滚动的视觉效果。



Left Mouse Down on Color_Bar sends that color to Status_Paint.BackColor (inner box) which in turn sends its BackColor value to object's FillColor Value.

Right Mouse Down on Color_Bar sends color to Status_OutLine (outer box) which in turn sends its BackColor value to the object's OutLineColor Value.

图 4.1 Artisan 的颜色调色板索引

4.2 ARTISAN.INI 文件

Color_INI 过程用一个 Windows API 过程将十六进制的颜色值写到 ARTISAN.INI 文件中，.INI 文件中也只写进了这些值。附带磁盘 16MISC 目录下有几个程序（如 VB-Clip 和 C-Clip）为了自己的目的将窗体和按钮的位置写入了 .INI 文件中，它们展示了几种不同的将这些值拼合起来以节省重复调用 .INI 文件次数的方法。

清单 4.5 从一个 .INI 文件中提取颜色值

```
Sub Color_INI ()
    ColorString = Space $ (144)
R = GetPrivateProfileString(KEY_NAME, KEY_ENTRY + "1",
    BAS_PALETTE1,ColorString, Len(ColorString), INI_FILENAME)
PaletteBarColors ColorString, 0
R = GetPrivateProfileString(KEY_NAME, KEY_ENTRY + "2",
    BAS_PALETTE2, ColorString, Len(ColorString), INI_FILENAME)
PaletteBarColors ColorString, 16
R = GetPrivateProfileString(KEY_NAME, KEY_ENTRY + "3",
    BAS_PALETTE3, ColorString, Len(ColorString), INI_FILENAME)
PaletteBarColors ColorString, 32
End Sub
```

ColorString：这是一个 144 字符的缓冲区，用来传递每一个颜色值串。

GetPrivateProfileString：这是一个 API 函数，该函数读取用户定义的 .INI 文件。

如果没有找到自定义的 .INI 文件，该函数就使用 ARTISAN7.BAS 文件中的默认颜色值，ARTISAN7.BAS 文件在附录 A 中。

PaletteBarColors：ColorBar 的值存在 Colors 数组中。此处关键的地方是数 0、16 和 32，这几个数用来计数三个单独的颜色值块。要搞清楚它是怎么执行的，最好的方法是用 F8 键或单步图标单步执行该过程。

4.3 用选定方框搜索一个对象

选定方框（marquee）是用指针工具画出的一个移动的点线方框。可以在一个对象周围画

一个点线方框以选定桌面的对象。

FindAllObjects 过程循环检查每一个可得到的对象位置坐标值，以找到桌面上的每一个边界方框。通常，总是只搜索选定区域，这将增加一个存储区域边界的数组。请记住，这只是一个示例程序。

本书中有几个不同的循环(Loop)过程，为了有助于更好地理解这些循环，我在本过程中按它们的长格式给出了这些循环。

另一个循环的例子可在 ZoomBar 事件 ZoomAll 中找到，ZoomAll 用一个两层循环找出所有的对象。另一个要考虑的过程是嵌套循环，有时它可能难以理解，但却能大大地提高搜索的速度。在附带磁盘的 16MISC 目录下 ArraySorter 程序中可以找到一个嵌套循环搜索的例子。

清单 4.6 FindAllObjects 过程在选定方框内查找对象

```
Sub FindAllObjects (X1%, Y1%, X2%, Y2%, X!, Y!)
CopyX1 = X1: CopyX2 = X2: CopyY1 = Y1: CopyY2 = Y2
If X2 > X1 And Y2 > Y1 Then X1 = CopyX1: X2 = CopyX2
    : Y1 = CopyY1: Y2 = CopyY2'NW >SE
If X2 > X1 And Y1 > Y2 Then X1 = CopyX1: X2 = CopyX2
    : Y1 = CopyY2: Y2 = CopyY1'SW >NE
If X1 > X2 And Y1 > Y2 Then X1 = CopyX2: X2 = CopyX1
    : Y1 = CopyY2: Y2 = CopyY1'SE >NW
If X1 > X2 And Y2 > Y1 Then X1 = CopyX2: X2 = CopyX1
    : Y1 = CopyY1: Y2 = CopyY2'NE >SW
I = Total
Do While I > 0
If E(I) >= W(I) And S(I) ,= N(I) Then West = W(I): East = E(I)
    : North = N(I): South = S(I)'NW >SE
If E(I) >= W(I) And N(I) >= S(I) Then West = W(I): East = E(I)
    : North = S(I): South = N(I)'SW >NE
If W(I) >= E(I) And N(I) >= S(I) Then West = E(I): East = W(I)
    : North = S(I): South = N(I)'SE >NW
If W(I) >= E(I) And S(I) >= N(I) Then West = E(I): East = W(I)
    : North = N(I): South = S(I)'NE >SW
If X1 <= West And Y1 <= North And X2 >= East And Y2 >= South
    Then : B = i: Handles: Nodes
I = I - 1
Loop
End Sub
```

FindAllObjects 过程可以找到当前桌面上所画的每一个对象，当用指针工具在一个对象的周围画出一个选定方框时，FindAllObjects 过程被用来选定该对象。画一个围绕多个对象的选定方框只会高亮显示最后所画的那个对象，因为本程序中没有一个组合或连接对象的过程。

CopyX1,CopyX2,CopyY1,CopyY2: 临时存放比较值的存储单元。

If X2>X1 And Y2>Y1 Then: 确定选定方框从哪里开始。下图中的白箭头显示了4个可能的起始点。

I=Total: 临时存储到目前为止所画对象的总数(Total)。

Do While：循环执行下面的代码，直到所有的对象都被找到为止。

If E(I)>=W(I) And S(I)>=N(I) Then：循环检查每一个边界方框位置，找出正确的 East(东)，West(西)，North(北)，South(南)顺序。

如果对象翻转了，则初始的 W，N，E，S 值被逆转是可能的，If/Then 语句搜索每一个可能的角度。如果前面列出的坐标在选定方框区域内，则图柄和节点都被置于该对象周围。

当画一个围绕一个对象的选定方框时，点 X1，Y1 和 X2，Y2 的坐标可能被反转或交换，这取决于开始画选定方框时拖动鼠标的方向。

在图 4.2 中，白色箭头显示了用指针工具画选定方框的四种可能的方法。X1 和 Y1 代表发生 MouseDown 事件时选定方框角点的坐标值。每一个选定方框都是沿对角线方向拖出直到松开按键为止(MouseUp 事件)，此时存储 X2，Y2 角点位置。

图 4.2 中的黑色箭头显示了一条单线是怎样被连续翻转的。每当翻转直线的边界方框时，西、北、东、南等屏幕位置也会翻转。

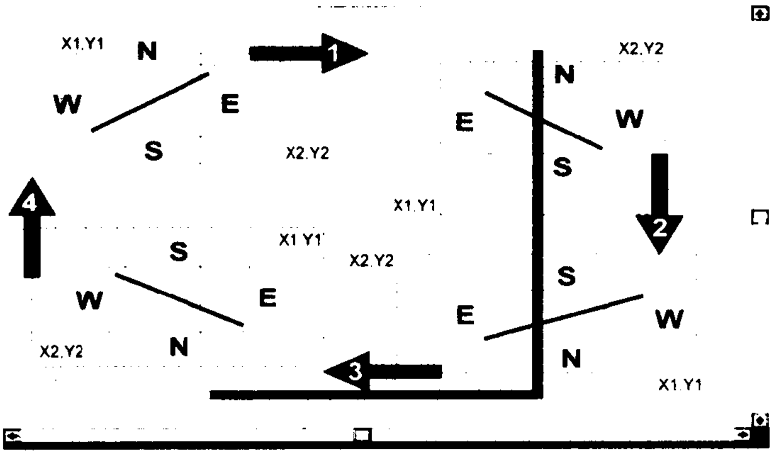


图 4.2 用指针工具画一个选定方框

4.4 在对象上单击鼠标

当用鼠标指针在一个对象上单击时，FindObjects 过程就高亮显示或选定该对象。但是此过程有一个缺点，那就是当在对象的边界方框内任何位置上单击鼠标时都会高亮显示该对象。在某些特定的程序中拖曳或选定一个对象时可能正需要这样。不过，如果你要通过对象的边界来选定一个对象，你可以参考第 15 章中的 ArtAPI 程序，该程序中有一个对象边框选定的例子。

为了找到某个对象的边界方框，FindObjects 过程搜索了 W，N，E，S 数组中所有的坐标值。Windows API 中有一个完成此功能的函数：PtInRect。

API 函数 PtInRect 可用来替代查找边界方框的过程，首先必须建立一个用户定义的类型结构，该结构应该能够存储每个边界方框的四个角，与本程序中所用的 W，N，E，S 数组类似。

下面是用户定义类型的一个例子：

```
Type Rect
```

```
    Left(1 to N) As Integer
    Top(1 to N) As Integer
    Right(1 to N) As Integer
    Bottom(1 to N) As Integer
End Type
```

第8章中的APIdraw程序给出了PtInRect函数查找Type Rect边界方框的一个例子。

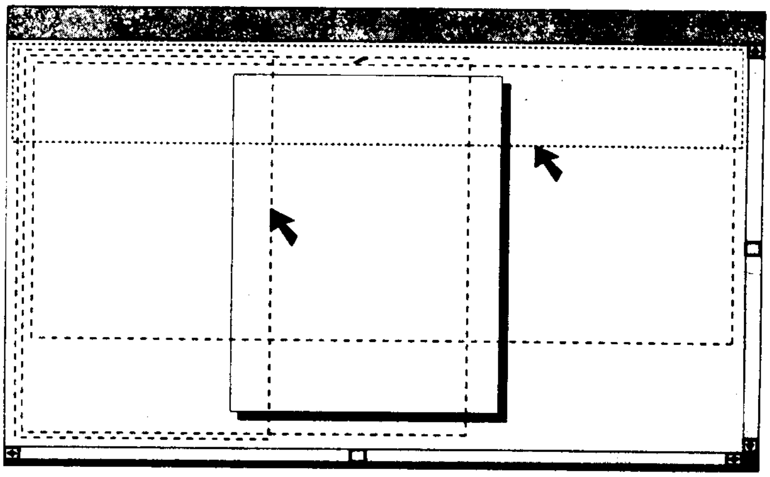


图4.3　标尺标线实际上是一些点线方框

清单4.7　查找位于鼠标位置下的边界方框

```
Sub Find_Object (X As Single, Y As Single)
R = TGL
Do While R > 0 '''Find GuideLines
If X < (GuideX(R) + 30) And X > (GuideX(R) - 30) Then
    ArtWork.DrawMode = Xor_Pen: ArtWork.DrawStyle = Dot
    ArtWork.AutoRedraw = True
    ArtWork.Line (-5000, -5000)-(GuideX(R), 24000), YELLOW, B
    ArtWork.AutoRedraw = False: X = -5000: SideRuler = True: V = R
    Exit Sub
End If
If Y < (GuideY(R) + 30) And Y > (GuideY(R) - 30) Then
    ArtWork.DrawMode = Xor_Pen: ArtWork.DrawStyle = Dot
    ArtWork.AutoRedraw = True
    ArtWork.Line (-5000, -5000)-(24000, GuideY(R)), YELLOW, B
    ArtWork.AutoRedraw = False: Y = -5000: TopRuler = True: V = R
    Exit Sub
End If
R = R - 1
Loop
''''''''''Find Bounding Box
I = Total
Do While I > 0
If S(I) >= N(I) And E(I) >= W(I) Then West = W(I): East = E(I)
    : North = N(I): South = S(I) 'NORMAL
If N(I) >= S(I) And W(I) >= E(I) Then West = E(I): East = W(I)
    : North = S(I): South = N(I) 'DIAGONAL
```

```
If N(I) >= S(I) And W(I) <= E(I) Then West = W(I): East = E(I)
     : North = S(I): South = N(I) 'top/bottom
If S(I) >= N(I) And W(I) >= E(I) Then West = E(I): East = W(I)
     : North = N(I): South = S(I) 'side to side
If X >= West And X <= East And Y >= North And Y <= South
     Then : Exit Do
I = I - 1
   Loop
If I = 0 Then PtInRectBasic = False: Exit Sub
      B = I: PtInRectBasic = True
      StatusLabel.Caption = Format $ (B)
      Artisan.StatusColor.BackColor = OutLineColor(B)
      Artisan.StatusPaint.BackColor = PaintColor(B)
      Nodes
End Sub
```

在 Find_Object 过程中，如果在标线上单击了鼠标(见图 4.3)，则该标线将被擦掉，以便移到别的位置。

点线标线实际上都是一些点线方框。如果能看到 ArtWork 绘图区域最外面的边(它们是隐藏的)，则它们(标线)看上去就与图 4.3 中所示的标线类似。

R=TGL:R 等于目前为止所画标线的总数。

Do While:循环处理到目前为止所画的全部标线。

If GuideX(R)=X Then:GuideX(索引号)存储点线标线的 X 坐标。这与存储边界方框位置方法相似。R 变量代表保存标线 X 坐标的索引号。在循环(Loop)中，如果发现鼠标落在 X 位置上，则执行 If/Then 表达内式的语句代码，直至 Exit Sub 语句。

ArtWork.DrawMode:正准备移动一条标线，所以将 DrawMode 改为 Xor，并开始绘制点线(Dot DrawStyle)。

ArtWork.AutoRedraw:必须擦除当前准备移动的标线。

ArtWork.Line:在原来的直线上再画一条新线，以擦去原有的线。

ArtWork.AutoRedraw:已经擦去了直线，所以现在关闭 AutoRedraw。

X=0:将 X 坐标设为 0，当按住鼠标按钮并移动时画出直线。

SideRuler=True:变量 SideRuler 标记为真，这样 ArtWork 的 MouseMove 事件就会用 GuideLines 过程执行画图操作。

Exit Sub:End If:任务已经完成，退出本过程。

下面的语句代码与 FindAllObjects 过程相似，不同的是接近代码底部的 If 语句。在循环检查所有可能的边界方框位置后，计算鼠标的 X,Y 位置是否在任何对象边界方框内。非常值得注意的是，它所找到的实际上是边界方框，而不是对象形状。例如在一个圆的外面单击也会高亮显示该对象，因为该圆在一个矩形的边界方框内(见图 4.4)。

B=I:如果本过程找到了一个对象，则当前边界方框的编号用 B 来表示，它就是指定给该对象的索引号。

PtInRectBasic:细节请参见清单 4.27 中的 MoveObject 过程。

StatusLabel:在标题框控制对象中显示对象的索引号。

ArtWork. Status：在 Status Area（状态区）中更新对象的当前颜色。

Nodes：刚刚高亮显示了一个边界方框，所以显示出该对象的节点并退出。

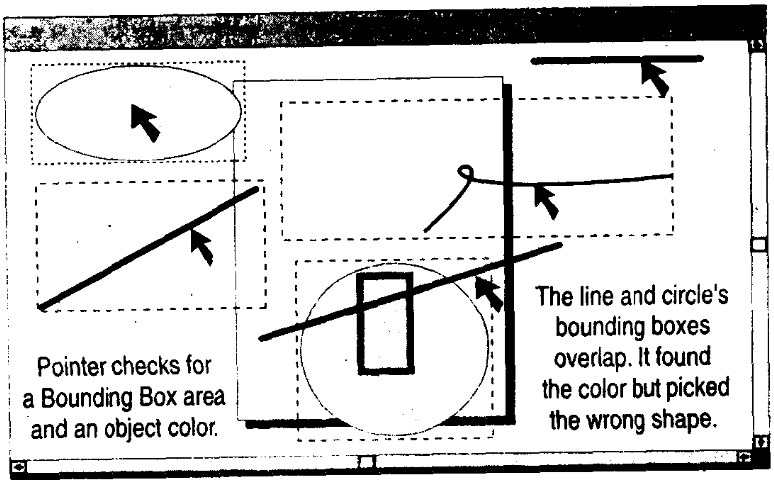

Pointer checks for a Bounding Box area and an object color.
The line and circle's bounding boxes overlap. It found the color but picked the wrong shape.

图 4.4　查找边界方框位置

4.5　绘制页标线

在桌面应用程序中，标线被用来通过观察对齐桌面上的文字或者对象。设置标线之后，将文字基线或者图形对象的边缘贴近或靠住点标线，即可将它们对齐。本程序不包括“贴近标线”功能，但浏览一下清单 4.7 中 Find_Object 过程的标线段就基本上可以理解是怎样执行的。FindObject 过程能够找到某条标线的坐标。然后在 ArtWork 图片框上发生 MouseMove 事件时就可以用相似的过程将对象的边界方框贴近标线的 X，Y 坐标。

清单 4.8　拖动任一标尺上的标线

```
Sub GuideLines (Button As Integer, X As Single, Y As Single)
ArtWork.DrawStyle = 2: ArtWork.DrawMode = 7
If SideRuler = True Then 'Side Ruler
    ArtWork.Line (-5000, -5000)-(X2, Y2), YELLOW, B
    X2 = X
    Y2 = 40000
    ArtWork.Line (-5000, -5000)-(X2, Y2), YELLOW, B
End If
If TopRuler = True Then 'Top Ruler
    ArtWork.Line (-5000, -5000)-(X2, Y2), YELLOW, B
    X2 = 40000
    Y2 = Y
    ArtWork.Line (-5000, -5000)-(X2, Y2), YELLOW, B
End If
End Sub
```

GuideLine 过程画出从侧边标尺（Ruler(0)）或顶部标尺（Ruler(1)）拖出的点线。

ArtWork. DrawStyle：标线是点线，所以将该属性设成点线。

ArtDrawMode：标线可能经过任何对象，所以将 DrawMode 改为 Xor，这样桌面上的对象就不会被擦掉。

If SideRuler＝True Then：正在拖动侧边标尺上的标线。

附注：标线实际上是方框，所以用这种标准的绘制移动方框对象的形式画出一个点线方框。

If TopRuler＝True Then：正在拖动顶部标尺上的标线。

当拖动侧边标尺或顶部标尺上的标线时，我们用 40000 作为 Y2 的值，这样可以保证在使用缩放（zoom）工具时标尺（实际上是一个点线方框）仍然可见。如果 Y2 的值太小，在滚动时，就可以看到点线方框的实际边界会出现在桌面区域内。如果该值太大，则可能因为超出了变量的数据极限而产生溢出错误。

也可用 Visual Basic 的 Line 方法（不带 Box 选项）来创建移动的标线，但是当直线很长时，如果要移动标线，则 Line 方法重画该标线的速度很慢。

4.6 定位对象周围的拉伸图柄

下面的几页讨论图柄。必须记住的是，有若干种不同类型的图柄事件和过程。下面是对每一种图柄事件或过程的简短描述：

Handle：它是一个图片框控制对象，也就是当一个对象被选定时，你在屏幕上看到的红色方框。该图柄（Handle）是一个有索引号的控制对象，它包含 8 个图片框，编号分别从 0 至 7。

Handles：这是一个过程，它在一个对象边界方框的周围放置 8 个小图片框（本程序中，图柄是红色的）。

HandlesReverse：这是一个过程，该过程在一个被翻转的对象边界方框周围放置 8 个小图片框。

BezHandle1：这是一个图片框控制对象，它仅用于移动 Bézier 曲线。

BezHandle2：这是一个图片框控制对象，它仅用于移动 Bézier 曲线。

附注：如图 4.5 所示，图柄从边界方框的角点位置偏移出来。这些位置（用 W(B)，N(B)，E(B)，S(B)存储）也等于每一个角上的白色节点的中心位置。白色节点宽度为 90 twip，所以，每个节点的中心在 45 twip 处。位于边界方框左半部分的图柄位置是用它们的宽度（120 twip）加上 45 twip 的偏移来设置的。位于边界方框左边线上的图柄只要使用偏移值 45 twip 即可。

在 Artisan 程序中，放在每个对象周围的 8 个小红色控制对象就是 Handle 图片框。当显示在屏幕上时，它们从对象的边界方框边缘稍稍偏出，这样使得在处理较小的对象时对象更易于观看，图柄更易于捕捉。

因为边界方框可能被翻转，所以坐标也可能被交换。鉴于此，需要有两个过程来放置这 8 个红色图柄框。Handles 过程用于正常的偏移处理，而当边界方框坐标被翻转时则使用 HandlesReverse 过程。

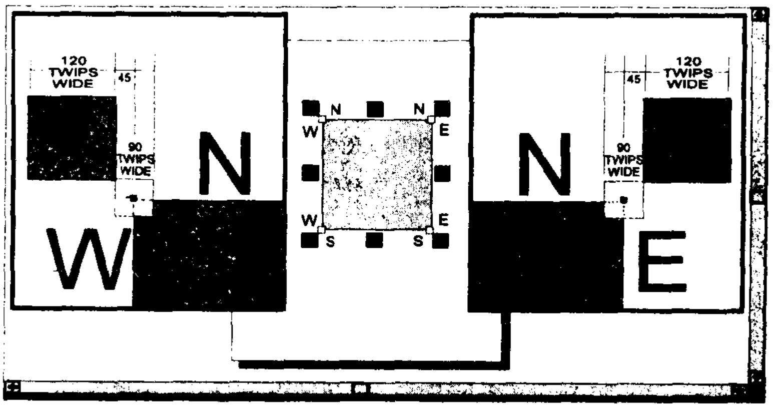


图 4.5　图柄的大小

清单 4.9　在对象周围放置 8 个拉伸图柄

```
Sub Handles ()
BezNode1.Visible = False: BezNode2.Visible = False
Shape1.Visible = False: Shape2.Visible = False
If N(B) > S(B) And W(B) < E(B) Or S(B) > N(B) And W(B) > E(B)
    Then HandlesReversed: Exit Sub
If S(B) > N(B) And E(B) > W(B) Then AA = -120: BB = 88
If N(B) > S(B) And W(B) > E(B) Then AA = 88: BB = -120
OSH = 44 + NodeZoom 'OFFSET Handles
    WC = W(B) - OSH: NC = N(B) - OSH: EC = E(B) - OSH: SC = S(B) - OSH
    MH = (WC \ 2 + EC \ 2)
    MV = (NC \ 2 + SC \ 2)
Handle(0).Left = WC + AA
Handle(0).Top = NC + AA
Handle(1).Left = MH
Handle(1).Top = NC + AA
Handle(2).Left = EC + BB
Handle(2).Top = NC + AA
Handle(3).Left = EC + BB
Handle(3).Top = MV
Handle(4).Left = EC + BB
Handle(4).Top = SC + BB
Handle(5).Left = MH
Handle(5).Top = SC + BB
Handle(6).Left = WC + AA
Handle(6).Top = SC + BB
Handle(7).Left = WC + AA
Handle(7).Top = MV
For R = 0 To 7
    Handle(R).ScaleLeft = Handle(R).Left
    Handle(R).ScaleTop = Handle(R).Top
    Handle(R).ScaleWidth = Handle(R).Width + Zoom
    Handle(R).ScaleHeight = Handle(R).Height + Zoom
    Handle(R).Visible = True
Next R
End Sub
```

BezNode1，BezNode2：如果这两个 Bézier 曲线控制对象是可见的，则隐藏起来。

Shape1，Shape2：如果这两个 Bézier 曲线控制对象是可见的，则隐藏起来。

If 语句：如果该对象先前被翻转了，则其边界方框的坐标也被逆置。调用 HandleReverse 过程在逆置的坐标值上加上偏移量，并退出本过程。

在某些页面布局程序中，拉伸图柄被用来改变对象的大小；Visual Basic 程序将图柄直接放在对象的边界方框上。简单地把图柄放在角点位置，并将每个角除以 2 就可以完成这一操作。必须注意的是，这种方法可能使图柄妨碍观看对象的边框。

If S(B)>N(B) And E(B)>W(B) Then：对象边界方框最初绘制时的方向决定了正确的 S，N，E，W 值，120 和 88 分别是图柄(Handle)和节点(Node)的宽度尺寸。Handle 图片框用 Left 属性来定位。例如，将图柄放在边界方框的右边就意味着只需要 44 twip 的偏移。

OSH＝44＋NodeZoom：每个节点宽 88 twip，高 88 twip，因此，值 44 表示每个节点的中心。加上偏移值 44 twip(距离)就可以将每个图柄靠近相邻的节点放置。NodeZoom 是当用缩放(zoom)工具扩大或缩小视图区域时所需的额外偏移(距离)值。

WC，NC，EC，SC：西角点，北角点，等等。若要放置水平方向上的中间图柄及垂直方向上的中间图柄，则必须如下所示的那样除以从每个图柄中心(OSH)开始的角点图柄坐标：

NH＝(WC\2＋EC\2)：

中间水平图柄＝西角点\2＋东角点\2。

MV＝(NC\2＋SC\2)：

中间垂直图柄＝北角点\2＋南角点\2。

Handle(索引号).Left：移动每个图柄的左边位置。

Handle(索引号).Top：移动每个图柄的顶部位置。

缩写：

WC＝西角点图柄 W(B)
NC＝北角点图柄 N(B)
EC＝东角点图柄 E(B)
SC＝南角点图柄 S(B)
MH＝中间水平图柄
MV＝中间垂直图柄
AA＝－120 或 105 twips；BB＝105 或－120 twips
OSH＝偏移图柄；2SH＝缩放比例图柄

改变 ArtWork 桌面的 Scale(比例)属性可以模拟放大或缩小页面上对象的操作。

因为所有图形对象都是在缩放后重画的，所以它们是根据选定的比例按比例画出来的，但是 Handles，Nodes，BezHandles，BezNodes 和文字需要特殊的处理。现在在 Artwork 图片框控制对象内所使用的图片框控制对象都是独立的。因为桌面的作用像是这些控制对象的一个载体，所以它们必须单独重新设定比例。当控制对象被重新缩放后，就必须补偿代码内所有内嵌的偏移值(offset)。本过程中的 OSH 变量就是这方面的一个例子。

图柄和节点只对其内部进行重新缩放，而其高度和宽度还是保持原样。例如，重设比例后，每个图柄的偏移值(offset)就不一定是 44 twips 了。

Visual Basic 的 ScaleLeft，ScaleTop 属性对窗体及图片框的内部比例尺寸有如下的作用：

ScaleLeft 改变从对象的顶部左边界开始的内部水平最左端坐标值。ScaleTop 改变从对象顶部边界开始的内部垂直最顶端坐标值。这两个属性的默认值都是 0。

ScaleLeft 和 ScaleTop 可以和 ScaleWidth 及 ScaleHeight 一起使用，用以指定窗体或图片框控制对象的自定义坐标。

ScaleWidth 改变内部水平比例宽度，ScaleHeight 改变内部垂直比例高度。设定为正值使得坐标从上至下、从左至右递增；而设定为负值则使得坐标从下至上，从右至左递增。

窗体或图片框的 BorderWidth 及 BorderStyle 属性影响其最终的内部尺寸。边界宽度(BorderWidth)减少内部可用的比例空间。

4.7　逆置翻转对象的图柄位置

只有对象的边界方框被翻转(无论任何方向)，才会从 Handles 过程中调用 HandleReverse 过程。若想看看不调用此过程会出现什么情况，则请注释掉(加上' 标记)Handles 过程头部的 If/Then 语句。结果是图柄沿着它们的 Left 和 Top 属性的相反位置对齐。

你可以设计一个始终要判断控制对象中心点的函数来绕过这个问题，你也可以要求 Microsoft 给控制对象加上一些诸如 Control. Right，Control. ScaleRight，Control. Center，Control. ScaleCenter 之类的属性，这样就可以省去置中一个窗体所必须做的全部多余的击键。

清单 4.10　在一个翻转的对象周围放置图柄

```
Sub HandlesReversed ()
If N(B) > S(B) And W(B) < E(B) Then AA = 88: BB = -120
If S(B) > N(B) And W(B) > E(B) Then AA = -120: BB = 88
OSH = 44 + NodeZoom 'OFFSET Handles
WC = W(B) - OSH: NC = N(B) - OSH: EC = E(B) - OSH: SC = S(B) - OSH
MH = (WC \ 2 + EC \ 2)
MV = (NC \ 2 + SC \ 2)
Handle(0).Left = WC + BB
Handle(0).Top = NC + AA
Handle(1).Left = MH
Handle(1).Top = NC + AA
Handle(2).Left = EC + AA
Handle(2).Top = NC + AA
Handle(3).Left = EC + AA
Handle(3).Top = MV
Handle(4).Left = EC + AA
Handle(4).Top = SC + BB
Handle(5).Left = MH
Handle(5).Top = SC + BB
Handle(6).Left = WC + BB
Handle(6).Top = SC + BB
Handle(7).Left = WC + BB
Handle(7).Top = MV
```

```
For R = 0 To 7
    Handle(R). ScaleLeft = Handle(R). Left
    Handle(R). ScaleTop = Handle(R). Top
    Handle(R). ScaleWidth = Handle(R). Width + Zoom
    Handle(R). ScaleHeight = Handle(R) Height + Zoom
    Handle(R). Visible = True
Next R
End Sub
```

If N(B)>S(B) And W(B)<E(B) Then：仅逆置 AA 和 BB 的值。有关每个代码行的解释请参见 Handles 过程(清单 4.9)。

Handle(索引号). Left：AA 和 BB 的值互换。

Handle(索引号). Top：AA 和 BB 的值互换。

缩写：

WC＝西角点图柄 W(B)
NC＝北角点图柄 N(B)
EC＝东角点图柄 E(B)
SC＝南角点图柄 S(B)
MH＝中间水平图柄
MV＝中间垂直图柄
AA＝－120 或 105 twips；BB＝105 或－120 twips
OSH＝偏移图柄；ZSH＝缩放比例图柄

人工绘制图柄

本章主要讨论将 Visual Basic 的图片框控制对象用作 Handles(图柄)和 Nodes(节点)。虽然这些控制对象加快了编程速度，但还是有个缺点，它们在桌面上显示起来很慢。一种替换的方法是自己画这些图柄，并花费相当多的时间将这些代码与 Artwork 桌面的鼠标事件连接起来。另一种方法是使用 Image(图象)控制对象，第 15 章中"ArtAPI 程序的简介"一节中将解释 Image 控制对象。

前面的代码例子展示了使用 Visual Basic 的 Line 语句(也就是删去了作为图柄的图片框控制对象)。这种方法可以省去令人头痛的重新设定控制对象比例的问题(每当控制对象被移动或使用缩放工具时)，遗憾的是，必须加入额外的代码，以便能够识别鼠标器事件。

下面是人工绘制图柄的例子。

```
Handle(0)=ArtWork. Line(WC+AA,NC+AA)-(WC+AA+105,NC+AA+105),RED,BF
Handle(1)=ArtWork. Line(MH,NC+AA)-(MH+105,NC+AA+105),RED,BF
Handle(2)=ArtWork. Line(EC+BB,NC+AA)-(EC+BB+105,NC+AA+105),RED,BF
Handle(3)=ArtWork. Line(EC+BB,MV)-(EC+BB+105,MV+105),RED,BF
Handle(4)=ArtWork. Line(EC+BB,SC+BB)-(EC+BB+105,SC+BB+105),RED,BF
Handle(5)=ArtWork. Line(MH,SC+BB)-(MH+105,SC+BB+105),RED,BF
Handle(6)=ArtWork. Line(WC+AA,SC+BB)-(WC+AA+105,SC+BB+105),RED,BF
Handle(7)=ArtWork. Line(WC+ +AA,MV)-(WC+AA+105,MV+105),RED,BF
```

4.8　准备一个待拉伸的对象

当一个对象被拉伸或被移至某个新位置时，该对象原来位置上的图象应该被擦掉。擦去该图象的方法之一是简单地用与桌面 BackColor（背景颜色）相同的颜色画一个填充的实心方框，然后重画可能位于该区域内的所有其他对象。

在清单 4.11 的 Handle_MouseDown 事件中，XX1，YY1，XX2，YY2 就是保存要拉伸或移动对象初始位置的变量。这些值被 UpdateObjects 过程（清单 4.12）中的一条 Line 语句用来擦除该区域。

清单 4.11　设置拉伸图柄的可移动方框

```
Sub Handle_MouseDown (Index%, Button%, Shift%, X!, Y!)
If B = FlagText(B) Then Exit Sub
ArtWork.DrawMode = Xor_Pen
ArtWork.DrawWidth = 1
    N_H_Clear
    XX1 = W(B)
    YY1 = N(B)
    XX2 = E(B)
    YY2 = S(B)
ArtWork.Line (W(B), N(B))-(E(B), S(B)), YELLOW, B
End Sub
```

有 3 个 Handle 鼠标事件是用来拉伸边界方框的（向各个方向），不要将这些事件和 Handles 过程混为一谈，Handles 过程是用来在边界方框周围放置 Handle（图柄）图片框控制对象的。

清单 4.12　重画页面及其上的所有对象

```
Sub UpdateObjects ()
LastMove = B
ArtWork.DrawMode = Copy_Pen
ArtWork.DrawStyle = Solid
ArtWork.AutoRedraw = True
If Draw = True Then GoTo SkipErase
ArtWork.DrawWidth = 20
ArtWork.Line (W(B), N(B))-(E(B), S(B)), White, BF
ArtWork.Line (XX2, YY2)-(XX1, YY1), White, BF: B = 1
       UpdateDeskTop
Do While B < Total + 1
  ArtWork.DrawWidth = LineWidth(B)
  If LineValue(B) > 0 Then ArtWork.DrawStyle = 5 Else ArtWork.DrawStyle = Solid
SkipErase:
Select Case B
    Case FlagCircle(B)
       X1 = W(B): Y1 = N(B): X2 = E(B): Y2 = S(B)
       ArtWork.DrawWidth = LineWidth(B)
       GoodCircle = PerfectCircle(X1, Y1, X2, Y2)
       ArtWork.FillColor = PaintColor(B): ArtWork.FillStyle = FillValue(B)
```

```
        ArtWork.Circle (XCenter, YCenter), Radius, OutLineColor(B), , , Aspect
        ArtWork.FillColor = 0: ArtWork.FillStyle = 1
    Case FlagLine(B)
        ArtWork.Line (W(B), N(B))-(E(B), S(B)), OutLineColor(B)
    Case FlagBox(B)
        If FillValue(B) = 1 Then GoTo SkipFill
        ArtWork.Line (W(B), N(B))-(E(B), S(B)), PaintColor(B), BF
SkipFill:
        ArtWork.Line (W(B), N(B))-(E(B), S(B)), OutLineColor(B), B
    Case FlagBézier(B)
        p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
        h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
        ArtWork.ForeColor = OutLineColor(B)
        BezDraw
    Case FlagText(B)
        ArtWork.CurrentX = OFFSET(B): ArtWork.CurrentY = N(B)
        SaveScaleLeft = ArtWork.ScaleLeft
        ArtWork.ScaleLeft = -ArtWork.CurrentX + Scaling
        ArtWork.FontName = TextFont(B)
        ArtWork.ForeColor = PaintColor(B)
        ArtWork.FontSize = TextSize(B) * TextPoint
        ArtWork.FontBold = TextBold(B)
        ArtWork.FontItalic = TextItalic(B)
        wFlag = FlagAlign(B)
        TA = SetTextAlign(ArtWork.hDC, wFlag)
        ArtWork.Print Storedtext(B)
        ArtWork.ScaleLeft = SaveScaleLeft
End Select
If Draw = True Then Draw = False:
        ArtWork.AutoRedraw = False: B = LastMove: Exit Sub
    B = B + 1
Loop
B = LastMove
    ArtWork.AutoRedraw = False
    ArtWork.DrawWidth = 1
    ArtWork.DrawStyle = Solid
If DeleteObject = True Then Exit Sub
If CurrentTool = 1 Or CurrentTool = 2 Then Exit Sub
If SideRuler = True Or TopRuler = True Then Exit Sub
    Handles
    Nodes
End Sub
```

所有的对象都是位于其相应的边界方框内，拉伸一个对象时，实际上是拉伸其边界方框。过程 UpdateObjects（清单 4.12）重画拉伸后的实际对象。

ArtWork.DrawMode：在拉伸一个边界方框时，该边界方框可能会经过其他的对象，因此将 DrawMode 改为 Xor，这样在拉伸一个边界方框时，其他的对象就不会被擦掉。

ArtWork.DrawWidth：DrawWidth 设为 1。

N_H_Clear：调用此过程清除（隐藏）节点和图柄。

现在，必须存储边界方框的当前坐标，它们存在 W(B)，N(B)，E(B)和 S(B)中。用临时

变量单元 XX1，YY1，XX2，YY2 来保存这些坐标，然后将它们传递给 UpdateObjects 过程，这样，在边界方框被拉伸或被移到桌面上别的位置上后，UpdateObjects 过程就会擦除原来区域中的图象。

XX1＝W(B)：临时保存边界方框的西边位置。

YY1＝N(B)：临时保存边界方框的北边位置。

XX2＝E(B)：临时保存边界方框的东边位置。

YY2＝S(B)：临时保存边界方框的南边位置。

ArtWork. Line：用存储在索引数组 W(B)，N(B)，E(B)，S(B)中的西，北，东，南坐标位置画当前边界方框。当使用 Xor 模式时，Line 语句中所用的黄颜色看上去象是青色（浅蓝色）。

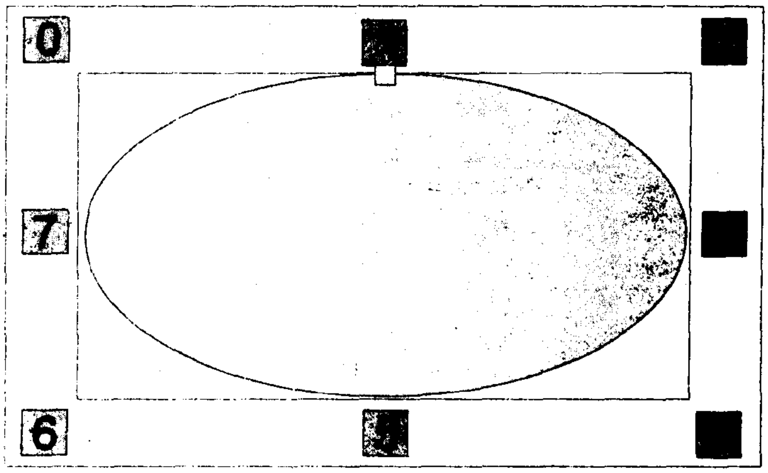


图 4.6　每个图柄的索引号

在图 4.6 中，每个图柄都是一个有索引的图片框控制。无论怎样拉伸或翻转边界方框，每个图柄都会呆在它恰当的位置上。Handles 过程（清单 4.9）用来将每个图柄移动到选定对象的恰当位置上。本程序中有 8 个被叫做 Handle（图柄）的有索引的图片框控制对象，它们的索引号从左上角开始顺时针方向分别为 0 至 7。

如图 4.7 所示，如果用图柄(3)将边界方框向右拉伸任意距离，MouseDown 事件首先就会隐藏住节点和图柄，原来的边界方框以及它内部的圆（见图 4.7）也必须擦除，原来的图象坐标存在 XX1，YY1，XX2 和 YY2 中。

接着，应该画出一个方框形状，而且当用鼠标拖曳图柄时，应该画出一个拉伸边界方框。

Handle_MouseMove 事件负责绘制拉伸边界方框，并在 StatusLabel 控制对象中显示出该边界方框的新的宽度和高度。

Handle_MouseUp 事件更新拉伸后的边界方框内的对象。如果该对象是一个方框、圆、直线或文字，则该过程调用 UpdateObjects 过程（清单 4.12）重画此对象；如果该对象是一条 Bézier 曲线，则先在 BezStretch 过程（清单 3.19）中计算出新的点，然后再传到 UpdateObjects 过程重画。

图 4.8 显示了一个内部有一个圆的边界方框的拉伸和翻转结果。拉伸从 Handle(0)（图柄 0）开始，向右下角拖曳。拉伸结束后，handles 过程如图所示重新定位各个拉伸图柄。

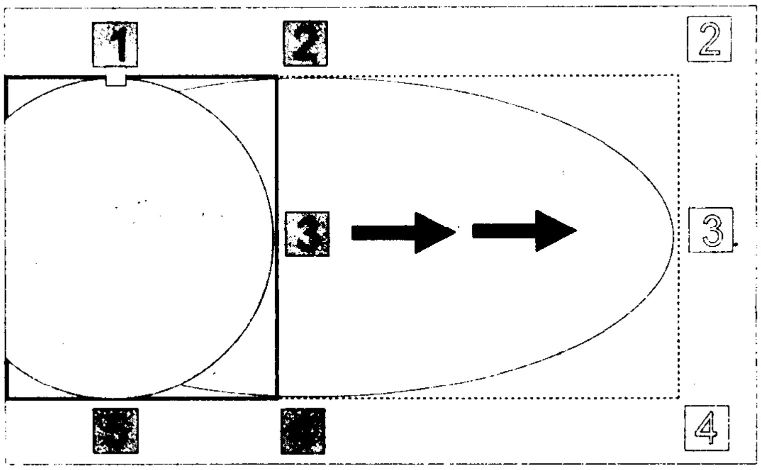
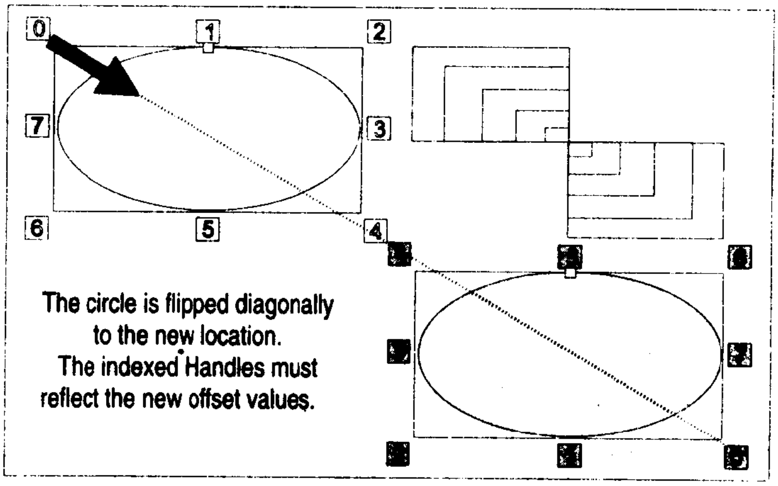


图 4.7　用图柄 3 拉伸边界方框



The circle is flipped diagonally to the new location. The indexed Handles must reflect the new offset values.

图 4.8　一个拉伸并翻转了的圆对象

4.9　拉伸一个对象

Handle_MouseMove（图柄）鼠标移动事件通过增大或减小坐标值 W(B)，N(B)，E(B)和/或 S(B)来拉伸对象的边界方框。

方框、圆及直线对象都是直接与其边界方框的边缘相连的，这使得在 UpdateObjects 过程（清单 4.12）中重画它们变得很简单。但是另一方面，Bézier 曲线是处于其边界方框之内的，所以应该用一个单独的过程来重定位曲线的四个坐标点。

组合或连接在一起的对象还需要有一个过程来建立拉伸后的边界方框。ArtAPI 程序中的 Poly_Stretch 用了一个过程来更新连接的多形状对象。

清单 4.13　拖曳并拉伸对象的边界方框

```
Sub Handle_MouseMove (Index%, Button%, Shift%, X!, Y!)
If B = FlagText(B) Then Exit Sub
```

```
Xstr = Format $ (X \ PixelX, "000")
Ystr = Format $ (Y \ PixelY, "000")
XYLabel.Caption = "[" + Xstr + " , " + Ystr + "]"
    RulerGuides Index, Button, Shift, X, Y
If Button Then
     ArtWork.Line (W(B), N(B))-(E(B), S(B)), YELLOW, B
     ArtWork.DrawStyle = Dot
Select Case Index
    Case 0
        Handle(0).MousePointer = 8
        W(B) = X
        N(B) = Y
    Case 1
        Handle(1).MousePointer = 7
        N(B) = Y
    Case 2
        Handle(2).MousePointer = 6
        E(B) = X
        N(B) = Y
    Case 3
        Handle(3).MousePointer = 9
        E(B) = X
    Case 4
        Handle(4).MousePointer = 8
        E(B) = X
        S(B) = Y
    Case 5
        Handle(5).MousePointer = 7
        S(B) = Y
    Case 6
        Handle(6).MousePointer = 6
        W(B) = X
        S(B) = Y
    Case 7
        Handle(7).MousePointer = 9
        W(B) = X
End Select
    ArtWork.Line (W(B), N(B))-(E(B), S(B)), YELLOW, B
End If
    BoxWidth = E(B) - W(B): BoxHeight = S(B) - N(B)
If Button Then StatusLabel.Caption = "Width " +
    Format $ (BoxWidth) + " " + "Height " + Format $ (BoxHeight)
End Sub
```

Xstr,Ystr：一个图柄就是一个独立的图片框控制，因此将其内部 X,Y 坐标送至 XYLabel 控制对象中显示。

RulerGuiders：反复调用 RulerGuiders 过程在每个标尺中画出页面位置线。

If Button Then：如果鼠标键被按住，则继续。

Select Case Index：执行等于当前选定的图柄索引号的 Case 语句。有 8 个图柄，索引号从 0 至 7。

ArtWork.Line：在第一条线上重新画一条线，以擦去该线。

ArtWork. DrawStyle＝Det：在拉伸时，边界方框变为点线矩形。

Case 0：鼠标指针西北、东南方向移动，西边和北边可以自由拉伸。

Case 1：鼠标指针北、南方向移动，北边可以自由拉伸。

Case 2：鼠标指针东北、西南方向移动，东边和西边可以自由拉伸。

Case 3：鼠标指针西、东方向移动，东边可以自由拉伸。

Case 4：鼠标指针西北、东南方向移动，东边和南边可以自由拉伸。

Case 5：鼠标指针北、南方向移动，南边可以自由拉伸。

Case 6：鼠标指针东北、西南方向移动，西边和南边可以自由拉伸。

Care 7：鼠标指针西、东方向移动，西边可以自由拉伸。

ArtWork. Line：在最后一条线上重画该线，与原有直线进行 Xor 操作。

BoxWidth，BoxHeight：方框被拉伸后的宽度和高度。这些值将显示在 StatusLabel 控制对象上。

表 4.1 是边界方框周围的 8 个图柄的位置明细表。图柄索引是从边界方框左上角开始按顺时针方向编号的。索引号与 Case 表达式对应。

表 4.1 边界方框上图柄的位置

索引	位 置
0：	左上
1：	中上
2：	右上
3：	右边
4：	右下
5：	中下
6：	左下
7：	左边

在图 4.9 中，灰色矩形外框位于 XX1，YY1，XX2，YY2 的位置。当该圆对象被拉伸到点线矩形区域所表示的新位置后，这个区域将被擦掉。原来的圆（灰色实心圆）在 UpdateObjects 过程中用下面的语句擦除：ArtWork. Line(XX2，YY2)－(XX1，YY1)，White，BF。在擦除时，边界被设成 20 个象素宽。

当一个对象被拉伸后，Handle_MouseUp 事件就调用 UpdateObjects 过程（清单 4.12）重画该对象。

清单 4.14 对象被拉伸后，更新其位置

```
Sub Handle_Mouseup (Index%, Button%, Shift%, X!, Y!)
If B = FlagText(B) Then Exit Sub
If FlagBézier(B) = B Then BezStretch Index, Button, Shift, X, Y
    UpdateObjects
    For R = 0 To 7
        Handle(R).MousePointer = 2
    Next R
```

```
End Sub
```

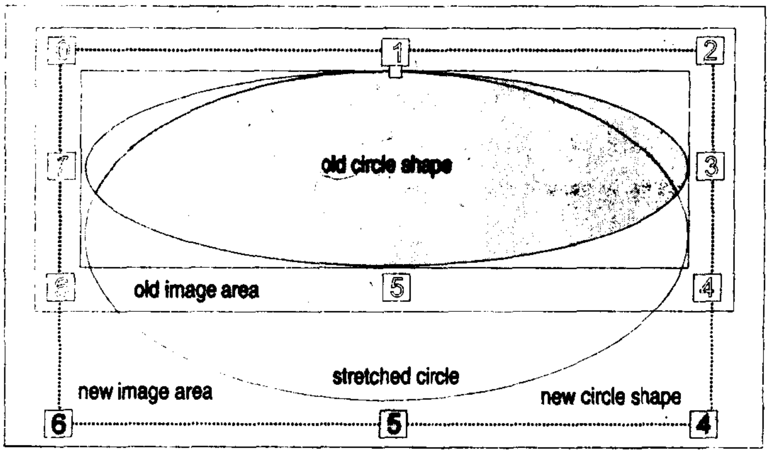


图 4.9　拉伸一个圆的示意图

如果当前对象是一条 Bézier 曲线，则在进行拉伸后，必须在调用 UpdateObjects 过程（清单 4.12）重画桌面上所有的对象之前更新该曲线的 4 个坐标点。全部 8 个图柄图片框的鼠标指针都被重设成十字线（鼠标指针 2）。

4.10　在桌面上滚动图形

HScroll 和 VScroll 滚动条用来对窗口进行垂直或水平移动，这是通过将 Artwork 图片框放在另一个图片框内来实现的。隐含的图片框叫做 PictureMask，该图片框不含任何代码。位于某个控制对象（该控制对象的作用类似于一个图片载体）内的控制对象可以在其隐含的载体上移至任何坐标位置。如果内部控制对象（Artwork）的任何部分在隐含载体（PictureMask）的实际视图区域之外，则 Visual Basic 就会自动地裁剪掉不可见的图象部分，这使得在 Visual Basic 内滚动任意类型的对象都变得很容易。

每个标尺内都有移动的点线，这些点线指示鼠标指针在桌面上的当前位置。这些标线是用 RulerGuides 过程（清单 4.15）绘制的。在离开 Artwork 图片框桌面区域时，点线标线就停留在鼠标离开前最后所在的位置。

清单 4.15　绘制标尺内部标线

```
Sub RulerGuides (Index%, Button%, Shift%, X!, Y!)
R0SL = Ruler(0).ScaleLeft 'Side Ruler
R0L = Ruler(0).Left - 5000: R0T = Ruler(0).Top - 5000
    R1ST = Ruler(1).ScaleTop'Top Ruler
    R1L = Ruler(1).ScaleLeft - 15: R1T = Ruler(1).ScaleTop - 15
Ruler(0).Line (R0L, R0T)-(X1Ruler, Y1Ruler), Cyan, B 'Vert
Ruler(1).Line (R1L, R1T)-(X2Ruler, Y2Ruler), Cyan, B 'Horz
   X1Ruler = R0SL + 500 'Side Ruler.Width
     Y1Ruler = Y
     X2Ruler = X
   Y2Ruler = R1ST + 500 'Top Ruler.Height
```

```
Ruler(0). Line (R0L, R0T)-(X1Ruler, Y1Ruler), Cyan, B 'Vert
Ruler(1). Line (R1L, R1T)-(X2Ruler, Y2Ruler), Cyan, B 'Horz
End Sub
```

必须有一种方法在滚动桌面时将 MouseMove 事件发送至每个滚动条控制对象，以擦除可见的标尺标线。不能给滚动条控制对象加上 MouseMove 事件，因为它们根本就没有 MouseMove 事件。

清单 4.16 水平滚动桌面

```
Sub HScroll_Change ()
ArtWork. Enabled = False
     Dim WM_MOUSEMOVE%
     WM_MOUSEMOVE% = &H200
API = SendMessageByNum(ArtWork. hWnd, WM_MOUSEMOVE%, 0, 0)
     ArtWork. Move -Hscroll. Value, ArtWork. Top
     Ruler(1). Move ArtWork. Left - 200, Ruler(1). Top
     BSM = (HScroll. Value * PixelX) * PerSW
End Sub
```

ArtWork. Enabled=False：使 Artwork 无效，这样就不会触发别的任何事件。

WM_MOUSEMOVE%=&H200：API 函数 SendMessage 有一个非常长的消息列表，它可以将这些消息发送给窗体或控制对象。这些消息存储在一个 API 库文件中，可以通过给 SendMessage 函数传递一个十六进制引用值来访问它们。若要给某个控制对象触发一个鼠标移动事件（该控制对象应该能接收此事件），十六进制值就应该是 &H200。

API=SendMessageByNum：API 消息可以通过文字或数字值发送。因为现在发送的是一个数字值，所以使用该过程的一个别名（SendMessageByNum），参数（数字值）被声明为非串值。SendMessage 函数必须在某个模块文件中声明。本程序中，该声明在 ARTISAN7. BAS 文件中，其格式如下：

```
Declare Function SendMessageByNum% Lib "User" Alias "SendMessage"
(ByVal hWnd%,ByVal wMsg%,ByVal wParam%,ByVal lParam%)
```

参考《Visual Basic 程序员指南》中的“调用 DLL 中的过程”，该章介绍了 API 函数调用及有关事项。

Artwork. Move：水平地移动（滚动）Artwork 桌面一定的距离，该距离由 HScroll 中的最小移动（small move）或最大移动（large move）属性指定。

Ruler(1). Move：标尺也必须移动以便与刚刚移动的 Artwork 对齐。另一种重新对齐标尺标记而不用移动标尺的方法在附带磁盘 14SCALE 目录下的 Ruler（标尺）程序中给出。Ruler 程序是第 14 章中 Scaletest 程序的一个副本，不过它采用了另一种缩放及对齐桌面和标尺的方法。

BSM=(HScroll. Value * PixelX) * PerSW：仅当加入了重新缩放桌面视图区域的缩放工具时才需要该值。它计算当前视图和缩放视图的比例差值。BSM 值用于 Ruler_Paint 事件中按新的比例重画标尺标记。BSM 是 Base Scale Movement（基准比例移动）的缩略

语。

清单 4.17　垂直滚动桌面

```
Sub VScroll_Change ()
ArtWork.Enabled = False
    Dim WM_MOUSEMOVE%
    WM_MOUSEMOVE% = &H200
API = SendMessageByNum(ArtWork.hWnd, WM_MOUSEMOVE%, 0, 0)
    ArtWork.Move ArtWork.Left, -Vscroll.Value
    Ruler(0).Move Ruler(0).Left, ArtWork.Top - 250
    SSM = (Vscroll.Value * PixelY) * PerSH
End Sub
```

上面的 VScroll_Change 事件（清单 4.17）中所用代码的细节请参见 HScroll_Change 事件（清单 4.16）。

4.11　改变对象边框的宽度

LineButtons 过程非常直接了当。若要改变某个对象边框的宽度，首先必须选定该对象，然后选定工具栏中的边框按钮（从 ToolButtons 控制对象顶部算起的第 7 个按钮）。

清单 4.18　LineButton_Down 改变一个对象的边框宽度

```
Sub LineButtons_MouseDown (Button%, Shift%, X!, Y!)
If Node(0).Visible = False
    Then LineButtons.Visible = False: Exit Sub
CurrentTool = 6: ToolButtons_Paint:
CurrentTool = LastToolUsed: ToolButtons_Paint
CurrentLine = X \ 32: LineButtons.Visible = False
Select Case CurrentLine
    Case LineColor
        ''''''''''''''''''''

     Case NoLine
         LineValue(B) = B
     Case HairLine
         ArtWork.DrawWidth = 1
     Case OnePt
         ArtWork.DrawWidth = 2
     Case TwoPt
         ArtWork.DrawWidth = 3
     Case FourPt
         ArtWork.DrawWidth = 5
     Case EightPt
         ArtWork.DrawWidth = 9
     Case TwelvePt
         ArtWork.DrawWidth = 13
End Select
LineWidth(B) = ArtWork.DrawWidth
If B = 0 Then Exit Sub
If Not CurrentLine = NoLine Then LineValue(B) = 0
```

```
    UpdateObjects
End Sub
```

If Node(0). Visible：首先检查 Node(0)是否可见，如果该节点不可见，说明还没有选定任何对象，所以退出本过程。

CurrentTool：使用与 ToolButtons 过程(清单 2.10)中相同的过程，将按钮反显，使得看上去像是被按了下去一样。ToolButton_Paint 事件(清单 2.11)会选定最后所用的画图工具。

Select Case CurrentLine：在 LineTools 栏中选定的按钮与以下的 Case 表达式中的某一个匹配。

Case：这些值都在 ARTISAN.BAS 文件中定义，简言之，它们就是从左至右排列的那些按钮。

Case LineColor：没有连接。你可以将它与 PaletteBar 窗体连接起来，并编写一小段代码以改变对象的边框颜色。现在这种情况下，你可以在 ColorBar 控制对象上单击鼠标右键以得到同样的结果。

CaseHairLine To Twelve Pt.：将文字以外的任意对象的边框宽度改为给定的值。

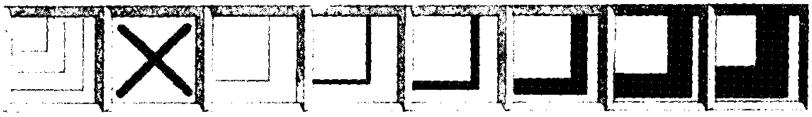
LineWidth(B)：存储所选定的线的宽度值。UpdateObjects 过程(清单 4.12)用来重画对象及其新边框。不要将此值设为零(0)，否则会产生运行错误。默认情况下，所有对象的宽度值(DrawWidth)均为 1。对象的边框是通过将 Artwork 的 DrawWidth 属性设置成存储在 LineWidth(对象编号)中的值来重画的。DrawWidth 不允许 0 值。

没有边框的对象被指定了不可见(invisible)属性，该属性是用标志变量 LineValue(对象编号)来设定的，如果该值为真，边框的 DrawStyle 属性就被设成不可见。

If B=0：B 代表对象的编号。启动时，B 等于 0，因为没有绘制任何对象。如果是这种情况，则退出此过程。

If Not CurrentLine：如果选定了除 NoValue 按钮以外的任何按钮，则将 LineValue(B)标记为可见直线，该直线将被实线方式重画。

LineButtons_MouseUp 事件将 Artwork 的绘图宽度设为 1 个象素，以准备在桌面上绘制新的对象。



清单 4.19 重新设置默认的桌面绘图宽度

```
Sub LineButtons_MouseUp(Button%,Shift%,X!,Y!)
    ArtWork.DrawWidth=1
End Sub
```

ArtWork.DrawWidth=1：将绘图宽度复位为默认的 1 个象素宽。

4.12　Artisan 程序的主菜单选项

大多数菜单项除了与语句 LastMenu＝Index 相连外，不再与任何别的过程代码相连。语句 LastMenu＝Index 是为了使弹出菜单能够显示出用户从主菜单标题上选定的最后一条菜单而设置的。任何时候，在桌面上单击鼠标右键就会激活菜单。

不含 Case 块的菜单就没有为该菜单项编制代码。本程序只是一个起点，并没有实现什么特殊的功能。

清单 4.20　将一个对象移到所有其他对象的前面或后面

```
Sub Menu_ArrangeSelection_Click (Index As Integer)
Select Case Index
    Case 0
        To_Front_Back Index
    Case 1
        To_Front_Back Index
End Select
End Sub
```

Case 0:调用 To_Front_Back 过程(清单 5.15)将选定的对象移到前面。

Case 1:调用 To_Front_Back 过程(清单 5.15)将选定的对象移到后面。

关于如何将一个对象移到所有其他对象的前面或后面的详细过程请参考第 5 章中的 To_Front_Back 过程。

第 15 章的 ArtAPI 程序中展示了一种更快的将一个对象移到所有其他对象前面或后面的方法。不过如果只需要排序少量的对象，则本章中所用的冒泡方法可能更适合。

4.13　从桌面上删除一个单独的对象

本程序通过将一个对象的全部属性都设为 0 来删去该对象，保存这些属性设置值的数组大小保持不变，占用的内存数量与其首次初始化时相同。

清单 4.21　从桌面上删除一个对象

```
Sub Menu_EditSelection_Click (Index As Integer)
Select Case Index
    Case 6 'DELETE
If B = 0 Then Exit Sub
If Node(0). Visible = True Then
    N_H_Clear
    FillValue(B) = 0: LineValue(B) = 0: OutLineColor(B) = 0
    PaintColor(B) = 0: LineWidth(B) = 1: FlagAlign(B) = 0
    OFFSET(B) = 0: TextSize(B) = 0: TextFont(B) = ""
    TextBold(B) = 0: TextItalic(B) = 0: CarriageCount(B) = 0
    Storedtext(B) = ""
    FlagBezier(B) = 0: FlagLine(B) = 0: FlagBox(B) = 0
    FlagCircle(B) = 0: FlagText(B) = 0
    p1X(B) = 0: p1Y(B) = 0: p2X(B) = 0: p2Y(B) = 0
```

```
        h1X(B) = 0: h1Y(B) = 0: h2X(B) = 0: h2Y(B) = 0
    DeleteObject = True
    XX1 = W(B): YY1 = N(B): XX2 = E(B): YY2 = S(B)
    W(B) = 0: N(B) = 0: E(B) = 0: S(B) = 0
        UpdateObjects
        DeleteObject = False
    End If
    End Select
    End Sub
```

N_H_Clear：隐藏显示在要被删除的对象周围的所有节点和图柄。

FillValue(B)=0:LineValue(B)=0，等等：总共有 26 个属性，但是，并不是所有的对象都需要这么多属性才能画出。因为这些存储属性值的数组大小不会改变，那么每个属性数组中与该对象编号对应的索引就含有值 0。将与待删除对象无关的数组中的属性索引设为 0 不会对桌面上的其他对象产生丝毫影响。

如果想要节省内存或存储太多的对象，就可能需要声明大小可变的属性数组。这种情况下，最好的办法是建立可以索引到单个组的多维数组。你可以将它设想成一个被分成若干个独立信息字段的数据程序，也可以考虑使用 Type 结构，此 Type 被声明为对象的名字(例如 Type Circle)，并在此 Type 结构中列出一个圆可能有的全部属性。

DeleteObject=True：这是一个标志，它告诉 UpdateObjects(清单 4.12)不要显示节点和图柄，因为该对象将不再存在。

XX1=W(B):YY1(N(B)：临时存放对象边界方框的位置，以便于 UpdateObjects 过程(清单 4.12)能够擦除该区域的图象。

W(B)=0:N(B)=0：因为该对象边界方框的位置已临时存储起来了，所以擦去边界方框的尺寸是安全的。

UpdateObject：重画桌面上的对象，减去刚删除的那一个。

4.14 为 File 菜单选项做准备

如果准备单击 File 菜单标题，则多半是要打开(open)、保存(save)或打印(print)图形。本例中，必须隐藏所有的节点、图柄和弹出工具。

清单 4.22 更新桌面上的节点和图柄

```
Sub Menu_File_Click ()
LastMenu = 0:N_H_Clear
If Txt.Caption > "" Then ReDrawText
    If BezHandle1.Visible = True Then
        ShowDotted = True: BezUpdate
        BezHandle1.Visible = False: BezHandle2.Visible = False
        BezNode1.Visible = False: BezNode2.Visible = False
        Shape1.Visible = False: Shape2.Visible = False
    End If
End Sub
```

If Txt.Caption> "" Then ReDrawText：在选定该菜单项时，有可能已经选定了一个文

字对象。这种情况下，就必须在保存或打印图象前重画文字。

如果 BezHandles1 是可见的，则已选定了一条 Bézier 曲线。此时就必须隐藏曲线的所有节点和图柄。

4.15　删除桌面上的全部对象

若要删除某个对象，就必须将该对象的全部属性都复位为 0。所有对象都给定了一个编号，从第一个对象开始编号为 1，直至最大值 1000。

重新指定任何对象的属性为 0 就是告诉本程序将该值指定给与该对象相关的索引号。这并不改变每个属性数组的大小，但将它们复位为启动时的初始值，也就是 0。

清单 4.23　清除所有数组值及对象

```
Sub Menu_FileSelection_Click (Index As Integer)
Select Case Index
  Case 0 'New
Erase FillValue, LineValue, OutLineColor, PaintColor, LineWidth
Erase FlagAlign, OFFSET, TextSize, TextFont, TextBold, TextItalic
Erase CarriageCount, Storedtext, W, E, N, S
Erase FlagBézier, FlagLine, FlagBox, FlagCircle, FlagText
Erase p1X, p1Y, p2X, p2Y, h1X, h1Y, h2X, h2Y
Erase GuideX, GuideY
p = 0: Total = 0: V = 0: B = 0: TGL = 0
     ArtWork.DrawWidth = 1
     ArtWork.DrawMode = 6: ArtWork.DrawStyle = 0
     ArtWork.AutoRedraw = True: ArtWork.Cls
     UpdateObjects
  Case 1 'OPEN
     mnuFileOpen
  Case 2 To 3
     mnuFileSave
  Case 4 'Print
     WinPrint
  Case 5 'EXIT
     End
End Select
End Sub
```

Case 0' New：该 Case 语句清除指定给所画的每个对象的全部属性。

有关从桌面上删除单个对象的更多的信息请参见清单 4.21。

Case 1 To 3：显示 CMDialog 命令对话框，然后就可以保存，另存或打开一个文件。

Case 4：已选定了菜单中的 Print（打印）选项。WinPrint 过程将所有对象打印到打印机上。WinPrint 被设置为仅在 300 dpi 的打印机上打印，但是你可以很容易地将代码改为按高于 300 dpi 的分辨率打印。

Case 5：选择退出本程序。

4.16 Color 菜单项选项

单击 Color 菜单项时，还会显示出一个菜单，该菜单还包括一个改变位于屏幕底部调色板内颜色的子菜单。

该子菜单选项在清单 4.25 的 Menu_Palette_Click 事件中列出。Color 菜单中仅剩的另一选项是隐藏标尺。

清单 4.24 编辑颜色调色板

```
Sub Menu_ColorSelection_Click (Index As Integer)
Select Case Index
     Case 2 'Show rulers
Menu_ColorSelection(2).Checked = Not Menu_ColorSelection(2).Checked
          Y1Ruler = 0: X2Ruler = 0
          RulerCross.Visible = Not RulerCross.Visible
          Ruler(0).Visible = Not Ruler(0).Visible
          Ruler(1).Visible = Not Ruler(1).Visible
End Select
End Sub
```

标尺的 Visible(可见)属性被 Not 运算符打开或关闭。在 Visual Basic 中打开或关闭某个东西最简单的方法就是用 Not 运算符。

计算机的一个内存位与电灯开关相似，位可能是 1(开)或 0(关)。因为一个位只能有两个值 0 或 1，所以位被称为一个二进制数。前缀 bi 的意思就是 2。

使用 Not 运算符后，值就变成它的反值：真变为假，或假变为真。如果值为空，则保持不变。

Not 运算符将任何变量的二进制位值反转，方法如下：(一个位只能是两个值中的一个，0 或 1)。

0 变为 1

1 变为 0

4.17 改变颜色调色板中的颜色

ColorPalette 窗体是 Visual Basic 示例程序 IconWorks 中的颜色调色板的改进版，另外又增加了一些功能，例如拖放颜色改变。可以用 ColorPalette 窗体中的 RGB 滚动条创建一种自定义颜色，然后将新颜色拖至主调色板显示区域。这样将会把该颜色存至第二调色板中，此调色板存在 ARTISAN.INI 文件中。

一旦改变了默认颜色调色板，则每次启动该程序时就会使用 ARTISAN.INI 调色板信息。若想要让程序在启动时使用默认调色板，则必须删去 ARTISAN.INI 文件。ColorPalette 窗体在附录 A 中进一步解释。

清单 4.25 重设默认颜色调色板

```
Sub Menu_Palette_Click (Index As Integer)
Select Case Index
    Case 0   'Show Color Palette
        ColorIndex = 0
        ColorPalette. Show
    Case 1   'Default Palette
        PaletteBarColors BAS_PALETTE1, 0
        PaletteBarColors BAS_PALETTE2, 16
        PaletteBarColors BAS_PALETTE3, 32
        Display_Color_Palette ColorBar
    Case 2
        Color_INI 'Custom Palette
        Display_Color_Palette ColorBar
    Case 3   'Save As Custom
        SaveColor_INI
End Select
End Sub
```

Select Case：根据选定了哪一个菜单项，执行下面相应的代码块：

Case 0：显示 ColorPalette 窗体。

ColorIndex=0：将颜色数组索引值设为第一个颜色位置。存在该数组中的全部颜色随后都被装入 ColorPalette 的显示调色板中。

Case 1：装入默认颜色调色板。

PaletteBarColors Bas_Palette1(2,3)：从 ARTISAN. BAS 文件中读取默认的十六进制颜色值。若想看看该文件，请参见附录 A。

Case 3：将拖进主调色板中的自定义颜色存入 ARTISAN. INI 文件中。

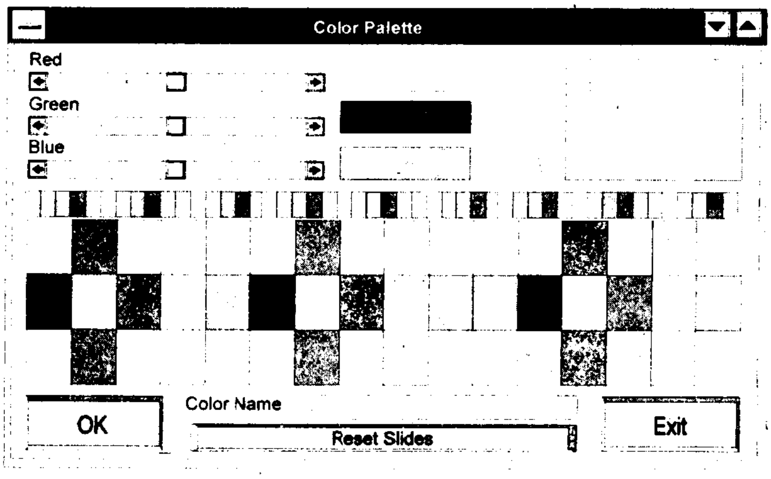


图 4.10 显示中的 ColorPalette 窗体

4.18 计算鼠标移动

当鼠标键被按住时，如果鼠标在桌面移动了一个给定的距离，则 MouseMove 函数返回

True(真),否则,返回 False(假)。如果用户由于某些偶然操作碰巧单击了桌面,就可以用此函数来终止不必要的对象绘制操作。

清单 4.26 判断鼠标是否移动了一个给定的距离

```
Function MoveMouse ()
If X1 > X2 + 45 Or X1 < X2 - 45 Or Y1 > Y2 + 45 Or Y1 < Y2 - 45 Then
    MoveMouse = True
Else
    MoveMouse = False
End If
End Function
```

If X1>X2+45 or X1<X2−45,等等,Then:X1,Y1 存储 Artwork 鼠标按下事件的当前 X,Y 位置。鼠标移动时,X2,Y2 存储 X,Y 值。当鼠标键按下时,鼠标必须移动预先设定的距离才能绘制对象,在采取任何操作之前,鼠标必须移动的距离是 45 twip 或 3 个象素(任何方向)。你可以将此值改为任意方向上的任意值。

4.19 在桌面上移动对象

MoveObject 过程移动一个颜色不等于 ArtWork 的 BackColor(背景色)的对象,该对象被置于鼠标指针处。

使用第 1 章的 MoveTest 程序中几种方法中的一种移动对象。对象移动是通过在其中边界方框区域内单击来实现的。若想通过仅单击对象边框形状来移动对象,请参考第 15 章中的 MoveObject 过程。

清单 4.27 用 on-the-fly 值移动一个对象

```
Sub MoveObject (Button%, Shift As Integer, X!, Y!)
If B = 0 Or PtInRectBasic = False Then Exit Sub
N_H_Clear
ArtWork.DrawMode = 7
ArtWork.DrawWidth = 1
'ArtWork.DrawStyle = Dot
ArtWork.MousePointer = 5
XCenter = X - ((W(B) \ 2) + (E(B)) \ 2)
YCenter = Y - ((N(B) \ 2) + (S(B)) \ 2)
ArtWork.Line (W(B), N(B))-(E(B), S(B)), YELLOW, B
W(B) = (W(B) - OFFSETx) + XCenter
N(B) = (N(B) - OFFSETy) + YCenter
E(B) = (E(B) - OFFSETx) + XCenter
S(B) = (S(B) - OFFSETy) + YCenter
If FlagBézier(B) = B Then
    p1X(B) = (p1X(B) - OFFSETx) + XCenter
    p1Y(B) = (p1Y(B) - OFFSETy) + YCenter
    p2X(B) = (p2X(B) - OFFSETx) + XCenter
    p2Y(B) = (p2Y(B) - OFFSETy) + YCenter
    h1X(B) = (h1X(B) - OFFSETx) + XCenter
    h1Y(B) = (h1Y(B) - OFFSETy) + YCenter
```

```
        h2X(B) = (h2X(B) - OFFSETx) + XCenter
        h2Y(B) = (h2Y(B) - OFFSETy) + YCenter
    End If
    If FlagText(B) = B Then OFFSET(B) = (OFFSET(B) - OFFSETx) + XCenter
       ArtWork.Line (W(B), N(B))-(E(B), S(B)), YELLOW, B
    End Sub
```

N_H_Clear：使所有节点和图柄都不可见。

ArtWork. DrawMode, Wide, Style：当拖动一个边界方框时，该边界方框就变成一个点线矩形，鼠标指针变成一个箭头(4)方向指针。

XCenter, YCenter：计算边界方框的中心。

W(B)=formula：边界方框的西边加上移动前方框的中心。

N(B), E(B), S(B)：北边，东边，南边加上中心位置。

If FlagBézier Then：公式与前面相同，只不过使用的是曲线的 4 个控制点。

If FalgText Then：将文字对齐方式设成新位置。索引数组 OFFSET(或文字对象编号)保存对齐偏移值，该值在 ReDrawText 过程(在清单 5.7 中解释)中定义。

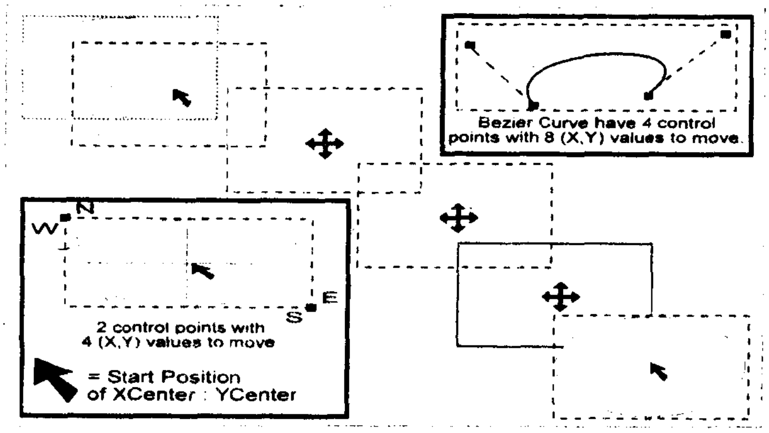

Bezier Curve have 4 control
points with 8 (X,Y) values to move.
N
W
E
S
2 control points with
4 (X,Y) values to move
= Start Position
of XCenter : YCenter

图 4.11　显示一个正被拖至新位置的圆

4.20　确定对象周围的节点的位置

Nodes 过程重新确定选定的对象内节点的位置。

节点是一些小图片框控制对象，它们显示在对象的直线及曲线点上。节点用来改变对象的形状或将选定的对象重新定位于新坐标上。

本程序中，使用节点只能重新确定直线或曲线端点的位置。在第 15 章的 ArtAPI 程序中，可以用节点改变一个多边形或曲线对象的形状。

清单 4.28　确定对象周围的节点的位置

```
Sub Nodes ()
OSN = 45 + NodeZoom 'OFFSET Nodes
Select Case B
    Case FlagLine(B)
```

```
        Node(0).Left = E(B) - OSN
        Node(0).Top = S(B) - OSN
        Node(2).Left = W(B) - OSN
        Node(2).Top = N(B) - OSN
        Node(0).Visible = True
        Node(1).Visible = False
        Node(2).Visible = True
        Node(3).Visible = False
    Case FlagBézier(B)
        Node(0).Left = p1X(B) - OSN
        Node(0).Top = p1Y(B) - OSN
        Node(1).Left = p2X(B) - OSN
        Node(1).Top = p2Y(B) - OSN
        Node(0).Visible = True
        Node(1).Visible = True
    Case FlagCircle(B)
        p = (W(B) + E(B)) \ 2
        Node(0).Left = P - OSN:Node(0).Top = N(B)-OSN
        Node(0).Visible = True:Node(1).Visible = False
        Node(2).Visible = False:Node(3).Visible = False
    Case FlagBox(B)
        Node(0).Left = E(B) - OSN:Node(0).Top = S(B) - OSN
        Node(1).Left = W(B) - OSN:Node(1).Top = S(B) - OSN
        Node(2).Left = W(B) - OSN:Node(2).Top = S(B) - OSN
        Node(3).Left = E(B) - OSN:Node(3).Top = S(B) - OSN
            For R = 0 To 3
                Node(R).Visible = True
            Next R

    Case FlagText(B)
        Node(0).Left = W(B) + Zoom:Node(0).Top= S(B) + Zoom
        Node(0).Visible = True
End Select
ReScale
End Sub
```

Nodes 过程将节点放在一个高亮显示对象的每一个端点或连接点处(不要与 BezNodes 相混淆,BezNodes 是用来移动 Bézier 曲线的独立控制对象)。

OSH:每个节点都是 90 twip 宽,90 twip 高,所以 45 正好代表其中心位置,它是正确对齐每个节点所需的偏移值。NodeZoom 是一个额外的偏移值,当用缩放工具重新缩放了视图区域时,要使用此值。

Select Case:B 是当前选定对象的编号。

Case FlagLine(B):两个节点,分别位于直线对象的两个端点。

Case FlagBézier(B):两个节点,分别位于曲线对象的两个端点。

Case FlagCircle(B):一个节点,位于圆上。

Case FlagBox(B):四个节点,分别位于方框的四个角。

Case FlagText(B):一个节点,位于第一个字母的下面。

用作节点的图片框控制对象只有四个。当对象被连接或每输入一个字符时,绘图程序都要使用许多节点。在运行时,可以加入更多的节点控制对象,但是要有一段更结构化的代码

来加以控制。有关连接复杂对象多个节点的例子请参见第13章中的节点连接程序。

下面是一个增加节点的例子：

```
Static Node Index as Integer
NodeIndex = NodeIndex + 1
Load PictureBox(NodeIndex)
PictureBox(NodeIndex).Visible = True
```

请注意，在默认情况下，新控制对象是不可见的。

NodeZoom：现在用来作为节点和图柄的是一些独立的图片框控制对象，它们位于Artwork图片框内。因为窗口的作用像是这些控制对象的载体，所以它们也必须重新设定比例（缩放）。

当控制对象被重新设定比例（缩放）后，必须在代码内加上额外的偏移补偿。本程序中的OSN变量就是这样的一个例子。图柄和节点只能对它们内部进行缩放，其高度和宽度都保持不变。例如在放大或缩小后，每个图柄的偏移就不一定是44 twip了，因为比例从原来的twip比例改成了用户定义的比例。缩放工具给变量NodeZoom加上一个额外的值以补尝新的比例尺寸。

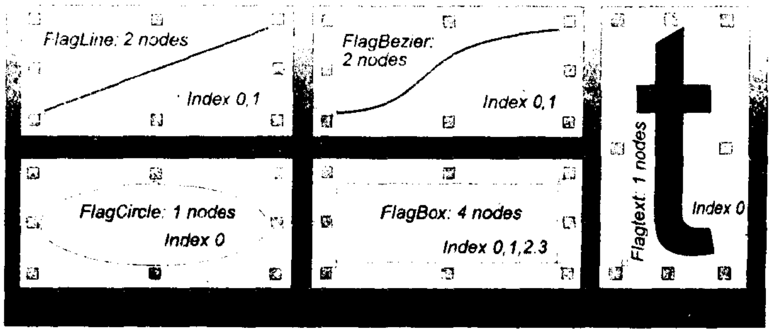

FlagLine: 2 nodes
Index 0,1
FlagBezier:
2 nodes
Index 0,1
FlagCircle: 1 nodes
Index 0
FlagBox: 4 nodes
Index 0,1,2.3
Flagtext: 1 nodes
Index 0

图4.12　对象周围节点的定位

4.21　从桌面上移去节点和图柄

N_H_Clear过程就如它所得到的结果那样简单，它只是隐藏节点和图柄（清单4.29）。

清单4.29　使节点和图柄不可见

```
Sub N_H_Clear ()
If B > 0 Then
    For R = 0 To 7
        Handle(R).Visible = False
    Next R
End If
  Node(0).Visible = False:Node(1).Visible = False
  Node(2).Visible = False:Node(3).Visible = False
End Sub
```

For R：循环计数器，从索引号 0 到索引号 7。

Handle(R)：隐藏全部 8 个图柄。

Node(0). Visible=False：隐藏全部 4 个节点。

4.22 擦去一个对象的填充颜色

标记为“×”的按钮就是 No_Fill 命令按钮，当一个填充了某种颜色的对象（方框或圆）被选定后，单击此按钮就会擦去填充颜色，使对象重新变为透明的。

清单 4.30 “×”按钮擦去对象上的填充颜色

```
Sub No_Fill_Click ()
If B = FlagLine(B) Or Node(0). Visible = False Then Exit Sub
     FillValue(B) = 1
     UpdateObjects
End Sub
```

If B=FlagLine(B)：如果该对象没有填充，则退出此事件。

FillValue(B)=1：将该对象的 FillValue 变量标志改为 1，这样就会在 UpdateObjects 过程（清单 4.12）中将 ArtAPI 的 FillStyle 改为透明（5）。

UpdateObjects：重画桌面上的所有对象。

4.23 本章小结

本章中比较好的部分就是讲述对象周围的 8 个图柄控制对象的那部分，这 8 个图柄是在对象画好后放置的。你可以利用这 8 个图片框控制对象内部固有的鼠标事件。不过，还必须考虑这样一个事实，那就是每当改变了 Artwork 图片框的内部比例后，就必须重新缩放位于其内的控制对象。解决这个问题的一种方法是直接在桌面上画出图柄形状，而不用图柄图片框。这就意味着必须加入一些额外的过程，以执行图柄鼠标过程所需的鼠标事件。这种类型的过程结构在用 C 一类的语言建立绘图包时很常见，C 语言在过程之间的跳转方面有更多的长处。在 Visual Basic 中，这将需要大量的工作和一个设计精美的过程结构，以使得所有可能的桌面鼠标过程从主 Artwork 图片框的鼠标事件中跳转出来。

在处理用作图柄的图片框时还应该考虑的是在对象周围实际放置图柄的那个过程。本程序中用了一个偏移值，以使图柄稍稍偏离正高亮显示的对象。如果对象被翻转了，还要用一个过程来逆置这些偏移。你可能想考虑用一个小的函数计算出图柄的内部中心点，然后，只用一个过程就把 8 个图柄放在一个不可见的矩形区域的周围。在 Windows 软件包中，这种方法很常用，它使用的是 Windows 的 API 函数 Region 和 Rect。本书不涉及 Region 和 Rect 函数的任何细节，因为函数太多。第 8 章中的 APIDraw 程序有一些有关使用得比较广泛的 API 函数的例子。

第 5 章　Artisan 程序——过程 N 至 Z

这是讨论 Artisan 程序的最后一章。在本章中，你将看到字母 N 到 Z 的 Artisan 程序过程，其中之一就是节点图片框鼠标事件，该事件用来重定位对象或改变对象的形状。

节点用来移动一条已经放置在 Artwork 上的直线。你可以给已画好的直线再加上其他的线。不过本程序中，任何另加的直线都被看成独立的对象，也就是说，它们没有连接或组合到一起作为一个单个的对象。而且节点鼠标事件只对与直线或曲线连接的节点起作用。

你可以再加一些代码以使节点处理其他的对象类型(例如在方框上生成一些圆角，或用圆的节点创建一个饼图或圆弧)。关于连接直线和曲线对象的例子，请参考第 13 章中的节点连接程序或第 15 章中的 ArtAPI 程序。

在专业的图形应用程序中，节点能够连接对象，将椭圆对象形状改为饼图或扇形形状，使方框的角成为圆角，以及按各种方式对齐文字字符，还可以增加或删除对象形状内的节点，凡此种种。

你可以将节点功能加到你自己的程序中。本书有几个比较小的图形应用程序，你可以从这些程序开始。不过，试图用 Visual Basic 来重新建立一个完全专业化的绘图包可能是异想天开、痴心妄想。你几乎完全不可能跟上专业绘图包中不断加进的新功能，Visual Basic 的执行速度与用 C 语言开发的应用程序比较起来还是要慢一些。必须记住，这里的 Artisan 程序是为了给你展示一些真实世界图形应用程序的基础知识，以便你能够改进你自己的应用程序，而无需过分依赖于 OCX 及 VBX 自定义控制对象。

5.1　通过节点移动一个对象

在 Artisan 程序中，用节点图片框控制对象只能够移动直线对象。在第 15 章的 ArtAPI 程序中，则可以移动用直线或曲线连接在一起的复杂对象。

当选定了 Bézier 工具后，与直线对象相连接的两个节点都可以用来将直线的任何一端移到一个新位置。当用这种方法画连续的直线时，这些直线不会相互连接起来，而是被当作独立的对象。第 13 章中的 Join Nodes 程序演示了如何连接直线对象。

清单 5.1　Node_MouseDown 擦除原有的直线图象

```
Sub Node_MouseDown (Index%, Button%, Shift%, X!, Y!)
If Not FlagLine(B) = B And Not FlagBézier(B) = B Then Exit Sub
    OSN = 45 + NodeZoom
    Node(0).Visible = False: Node(2).Visible = False
ArtWork.DrawMode = Inverse
Select Case CurrentTool
     Case BézierTool
         ArtWork.DrawMode = 13
         ArtWork.AutoRedraw = True
         ArtWork.Line (W(B), N(B))-(E(B), S(B)), White
```

```
      ArtWork. AutoRedraw = False
      ArtWork. DrawMode = Inverse
      ArtWork. Line (W(B), N(B))-(E(B), S(B)), White
      If Index = 2 Then
         X1 = W(B): Y1 = N(B)
         X2 = E(B): Y2 = S(B)
         W(B) = E(B): N(B) = S(B)
      ElseIf Index = 0 Then
         X1 = E(B): Y1 = S(B)
         X2 = W(B): Y2 = N(B)
      End If
XX2 = X2: YY2 = Y2: XX1 = X1: YY1 = Y1
  Case LineTool
   If Index = 0 Then X1 = Node(0).Left + [IC:CCC]OSN: Y1 = Node(0).Top + OSN
   If Index = 2 Then X1 = Node(2).Left + [IC:CCC]OSN: Y1 = Node(2).Top + OSN
      X2 = X1: Y2 = Y1
   End Select
End Sub
```

If Not FlagLine(B)=B And Not FlagBézier(B)=B Then:如果当前选定的对象不是直线对象,则退出 MouseDown 事件。

OSN:此变量代表对齐节点中心所需的偏移。NodeZoom 是在使用了缩放工具时所需的额外偏移值。

Node(0). Visible:在移动一个直线段时,隐藏当前节点。

Node(2). Visible:在移动一个直线段时,隐藏当前节点。

Select Case CurrentTool:转至与当前选定工具相等的 Case 表达式。

Case BézierTool:当前选定的是 Bézier 工具。

If FlagBézier(B):如果当前选定的对象是显示一条 Bézier 曲线的图柄,则用 BezUpdate 过程显示出该 Bézier 曲线的节点及其连接点线。

If FlagLine(B)=B And Index=2:你也可以用 Bezier 工具将一条直线移至另一位置。画一条移动的直线必须从一个固定的起点开始,否则就会在该直线段的相反端开始。Nodes 过程(清单 4.26)指定了两个节点,分别位于直线的两端。本程序中用了 4 个节点控制对象。直线对象使用 Node(0)和 Node(2)。如果当前对象实际上是直线对象,而且当前选定的直线节点索引号为 2,则使用下面的坐标:

```
X1=W(B):Y1=N(B):X2=E(B):Y2=N(B)
```

W(B)=E(B):N(B)=S(B):逆置直线的起点和终点。

ElseIf:如果直线节点的索引号为 0,则使用正常直线坐标。

XX2,YY2,XX1,YY1:临时存储直线初始边界方框的位置,这样在移动了该直线的一个节点后就可以擦去直线原来的图象。

Case LineTool:当前在 ToolButtons 中选定的工具是直线工具。

If Index = 0:If Index 1:该直线节点索引号为 0 或者 2,新直线将从这两个索引节点开始。必须计算出当前节点的中心,以便将新直线的起点与当前直线的终点连接起来。

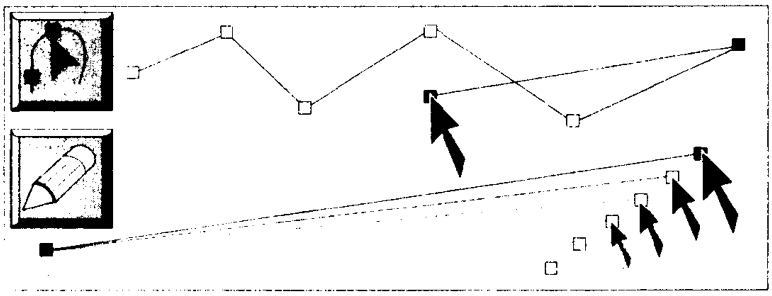


图 5.1　拖曳一个直线节点

5.2　移动直线对象的节点

当移动两个节点中的任何一个时，节点的鼠标移动事件就用 Line 语句重画该直线。

清单 5.2　移动直线时，Node_MouseMove 事件重画该直线

```
Sub Node_MouseMove (Index%, Button%,Shift%, X!, Y!)
Xstr = Format$ (X \ PIXELX, "000")
Ystr = Format$ (Y \ PIXELY, "000")
XYLabel.Caption = "[" + Xstr + " . " + Ystr + "]"
If Not FlagLine(B) = B And Not FlagBézier(B) = B Then Exit Sub
If Button Then
  Select Case CurrentTool
    Case BézierTool
      If FlagLine(B) = B Then
        ArtWork.Line (X1, Y1)-(X2, Y2)
        X1 = X: Y1 = Y
        ArtWork.Line (X1, Y1)-(X2, Y2)
      End If
    Case LineTool
      ArtWork.Line (X1, Y1)-(X2, Y2)
      X2 = X: Y2 = Y
      ArtWork.Line (X1, Y1)-(X2, Y2)
  End Select
End If
End Sub
```

If Not FlagLine(B)=B Then：当前选定的对象不是直线对象，因此退出鼠标移动事件。

If Button Then：如果鼠标键被按住，继续。

Select Case CurrentTool：转至与当前选定的工具对应的 Case 表达式。

Case BézierTool：当前从 ToolButtons 选定的是 Bézier 工具。

ArtWork. Line：使用标准公式画移动对象。

Case LineTool：当前从 ToolButtons 选定的是直线工具。

ArtWork. Line：使用标准公式画移动对象。

图 5.2 示出了用 LineTool（直线工具）从每个直线段的最后节点位置连续画直线的结果。这个五星形状并没有组合成一个单个的对象，而是由 10 个独立的直线对象构成。本程序并不连接或组合对象。

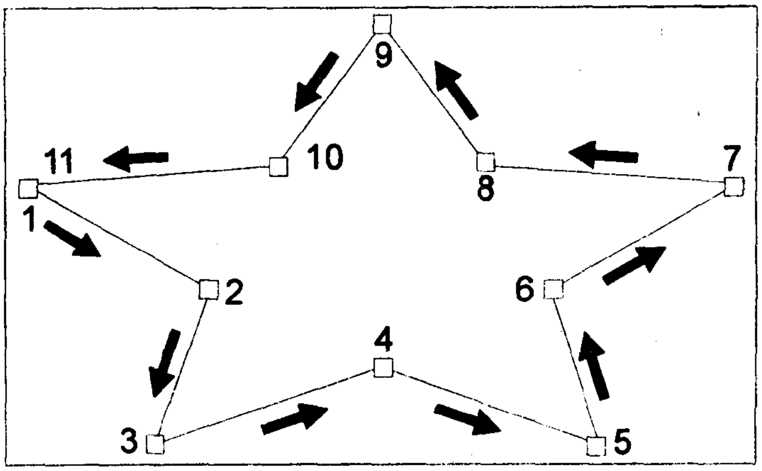


图 5.2　从一个节点起连续画直线对象

图 5.3 显示了用 BézierTool（Bézier 工具）将原先的五星形状位置中的一个节点重新定位后的结果。因为这个五星不是一个单个的对象，所以，必须选定构成该五星的 10 个直线段中的某一段，才能显示出该段的两个节点。

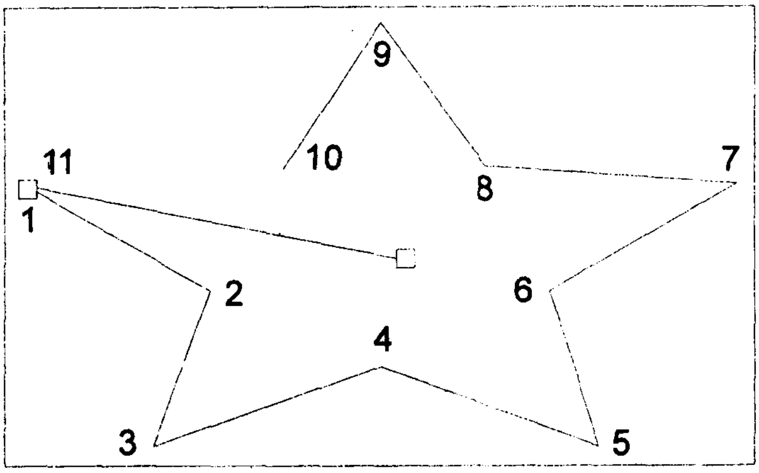


图 5.3　移动一个直线对象的节点

5.3　移动节点后更新直线对象

节点的 MouseUp（鼠标抬起）事件在移动了直线的两个节点中的任意一个后重新计算该直线的边界方框位置。

清单 5.3　Node_MouseUp 事件更新直线的坐标

```
Sub Node_MouseUp (Index%, Button%, Shift%, X!, Y!)
If Not FlagLine(B) = B And Not FlagBézier(B) = B Then Exit Sub
Select Case CurrentTool
    Case BézierTool
        W(B) = X1: N(B) = Y1: E(B) = X2: S(B) = Y2
        UpdateObjects
        Nodes
    Case LineTool
        ArtWork.DrawMode = Copy_Pen
        BoundingBox_Up X1, Y1, X2, Y2
        If B = Total Then Draw = True: UpdateObjects
    End Select
End Sub
```

If Not FlagLine(B)=B Then：当前选定的对象不是直线对象，所以，退出 MouseUp 事件。

Select Case CurrentTool：转至与当前选定工具相应的 Case 表达式。

Case BézierTool：**W(B)**，**N(B)**，**E(B)**，**S(B)**：存储该直线段的新边界方框位置。调用 UpdateObjects 过程重画该直线，然后再将该直线的节点重新定位于新的位置。

Case LineTool：当前从 ToolButtons 选定的是直线工具。

ArtWork. DrawMode：将绘图模式复位为正常模式。

BoundingBox_Up：生成了一个新直线对象，所以必须将对象总数增 1。BoundingBox_Up 过程（清单 2.20）完成增加对象计数所需的所有计算。

If B=Total：如果该直线是新对象，则只需重画该直线而无需重画桌面上的所有对象。将 Draw 变量标记为 True（真）就会使得 UpdateObjects 只画桌面上的新直线对象。

5.4　使用弹出式绘图工具绘图一个对象

PaintButtons 图片框包含了一个 Color Fill（颜色填充）按钮图象的位图。要将某个对象的填充颜色改为一种预设的颜色，则只须单击含有你所喜爱颜色图象填充按钮即可。代码与从 ToolButtons 图片框控制对象（清单 2.10）中选定一个工具类似。本程序中的所有按钮栏均采用按钮的位图图象，而不是 Visual Basic 的命令按钮（Command Button）。你可能想采用图象（image）控制对象，而不是图片框控制对象（图象控制对象更快，而且节省资源）。有关替代位图按钮图象的例子请参考附带磁盘 16MISC 目录下的 Buttons 程序。

清单 5.4　用某种颜色填充选定的对象

```
Sub PaintButtons_MouseDown (Button%, Shift%, X!, Y!)
If Node(0).Visible = False Then PaintButtons.Visible = False: Exit Sub
CurrentTool = 7: ToolButtons_Paint: CurrentTool = LastToolUsed: ToolButtons_Paint
```

```
CurrentBrush = X \ 32: PaintButtons.Visible = False

Select Case CurrentBrush
     Case Palette

         CMDialog1.Color = PaintColor(B)
         CMDialog1.Flags = &H2& Or &H1&
         CMDialog1.Action = 3
         PaintColor(B) = CMDialog1.Color: FillValue(B) = 0
     Case NoFill
         No_Fill_Click
     Case WhiteBrush
         PaintColor(B) = &HFFFFFF: FillValue(B) = 0
     Case Brush10
         PaintColor(B) = &HE5E5E5: FillValue(B) = 0
     Case Brush20
         PaintColor(B) = &HCCCCCC: FillValue(B) = 0
     Case Brush50
         PaintColor(B) = &H7F7F7F: FillValue(B) = 0
     Case Brush70
         PaintColor(B) = &H4C4C4C: FillValue(B) = 0
     Case BlackBrush
         PaintColor(B) = &H0: FillValue(B) = 0
End Select
    UpdateObjects
End Sub
```

关于怎样使得按钮看上去像是被按进去了的解释，请参考 ToolButton 鼠标器事件(清单 2.10)中所用的过程。

Select Case CurrentBrush：你可以按七个按钮中的任意一个。

Cases：用来填充对象的画笔颜色(填充颜色)。CurrentBrush(当前画笔)的名字(Brush10，Brush20，等等)都是一些全局常量，它们在 ARTISAN.BAS 文件中声明，该文件在附录 A 中。

Case Palette：显示 CMDialog 公共对话框控制对象。当 CMDialog 控制对象出现后，就可以从对话窗体中选择一种自定义的颜色来填充对象了。

CMDialog1：有关 CMDialog 控制对象的方法和属性值的说明，请参考《Visual Basic 语言指南》中的“公共对话框控制对象”和“过滤筛选器”。

Case NoFill：调用 No_Fill Click 事件(清单 4.30)擦去填充颜色。

所有其他情况：将该对象的颜色索引值改成一个新的十六进制颜色值(可以是你选择的任意值)，然后，将该对象的 FillValue 标志变量设为 0，UpdateObjects(清单 5.16)并根据该变量值用实(纯)颜色重画该对象。

5.5 从.INI 文件中抽取颜色值

PaletteBarColors 获取一组字符串(此例中是十六进制颜色值)，并将每一个值都赋给一个索引数组 Colors。该串包含 16 个十六进制的颜色值，这些颜色值紧挨着存放，每两个之间

用一个空格隔开。ColorString 的值在 ARTISAN.BAS 的声明段中(附录 A),并被划分成三个部分:BAS_PALETTE1,BAS_PALETTE2 和 BAS_PALETTE3。如果在 ARTISAN.INI 文件中定义了一个自定义的调色板,则用 ARTISAN.INI 中列出的自定义颜色值来定义调色板的颜色。

清单 5.5　PaletteBarColors 从一个 .INI 文件中抽取颜色值

```
Sub PaletteBarColors (ColorString $ ，FirstColor，RefreshPalette)
Start = 1
    For C = FirstColor To FirstColor + 15
        Colors(C) = Val(Mid $ (ColorString，Start，8))
        Start = Start + 9
    Next C
End Sub
```

C=:给抽取出的每个十六进制颜色值指定的索引号。

FirstColor=:从 Color_INI 过程(清单 4.5)传来的一个参数。Color_INI 过程传递的值有 0、16 或 32,它代表开始抽取的三组十六进制颜色值中每一组的插入点。FirstColor+15 说明每一组有 16 个十六进制颜色。当 Color_INI 过程最终调用 PaletteBarColors 过程三次以后,调色板中就会有 48 种颜色。

Color (C)=Val(Mid $ (ColorString $,Start,8)):数组 Colors 按索引号(0 至 47)存放全部 48 种颜色。每个十六进制颜色值都是 8 位数字长,所以 Visual Basic 的串函数 Mid $ 就可以从成组的数字串(如 BAS_PALETTE1)等中抽取出每一个十六进制颜色。

Start=Start+9:每个 8 位数字字符的十六进制颜色值都是由一个空格隔开的。每抽取出一种颜色后就将 Mid $ 函数中的起始位置增加 9 个字符位置,从而可以正确定位下一次颜色抽取位置。

5.6　创建正圆

PerfectCircle 函数也可以在 Visual Basic 的 IconWorks 例子中找到。在此处包括该例子的一个改进版本,因为它计算移动圆的半径时只使用了 Basic 的语句。不过可以用 API 函数 Ellipse 来替换掉此 PerfectCircle 函数。第 8 章中的 DrawAPI 程序给出了一个用 API 的 Ellipse 函数画移动的圆对象的例子。第 15 章中的 ArtAPI 程序也使用了 API 函数 Ellipse。

清单 5.6　计算一个移动的圆的半径

```
Function PerfectCircle (X1，Y1，X2，Y2)
Xleg = Abs(X2 - X1)
If Xleg <> 0 Then
    Yleg = Abs(Y2 - Y1)
    Aspect = Abs(Yleg / Xleg)
If Xleg > Yleg Then Radius = Xleg \ 2 Else Radius = Yleg \ 2
    XCenter = X1 + (X2 - X1) \ 2
    YCenter = Y1 + (Y2 - Y1) \ 2
End If
```

```
        PerfectCircle = Xleg
End Function
```

XLeg=：圆的边界方框宽度的绝对值。

YLeg=：圆的边界方框高度的绝对值。

Aspect=：绝对纵横比。如果此值等于1，则该圆就是一个正圆。

Radius=：边界方框宽或高的中心位置。

XCenter，YCenter=：圆的边界方框的中心点。

5.7 用 Visual Basic 在桌面上放置文字

ReDrawText 过程用来重画在 Txt 标题控制对象中或在 Text 窗体的文字框控制对象中输入的文字。ReDrawText 过程将文字串传到桌面上，并给文字指定一个对象编号。

ReDrawText 过程仅使用了标准的 Visual Basic 文字属性来绘制文字。因此，在滚动文字时就受到许多限制。Windows API 中的文字函数比 Visual Basic 更好。在第12章中有三个 API 文字程序的小例子，这些例子涉及 API 函数 TextOut 和 DrawText。

清单 5.7 在桌面上绘制文字对象

```
Sub ReDrawText ()
If txt.Caption > "" Then
Total = Total + 1: B = Total
CarriageCount(B) = crCounter: crCounter = 0
LineWidth(B) = ArtWork.DrawWidth
FlagText(B) = B: FillValue(B) = 1
    TL = txt.Left: TT = txt.Top
    TW = txt.Left + txt.Width
    TH = (txt.Top + txt.Height)
    Txt.Visible = False
TextSize(B) = txt.FontSize ' Size of Txt.Label Type
Storedtext(B) = RTrim(txt.Caption)
TextFont(B) = RTrim(ArtWork.FontName)
TextBold(B) = ArtWork.FontBold
TextItalic(B) = ArtWork.FontItalic
If Align = 0 Then FlagAlign(B) = TA_LEFT: OS = 0
If Align = 6 Then FlagAlign(B) = TA_CENTER: OS = (txt.Width / 2)
If Align = 2 Then FlagAlign(B) = TA_RIGHT: OS = txt.Width
''''''''''''''''''''''''''''''''''''''
    ArtWork.AutoRedraw = True
    ArtWork.CurrentX = TL
    OFFSET(B) = ArtWork.CurrentX
    ArtWork.CurrentY = TT
    SaveScaleLeft = ArtWork.ScaleLeft
    ArtWork.ScaleLeft = -ArtWork.CurrentX + Scaling
    ArtWork.ForeColor = PaintColor(B)
    ArtWork.FontName = TextFont(B)
    ArtWork.FontSize = TextSize(B)
```

```
        TextSize(B) = txt.FontSize / TextPoint
        ArtWork.FontBold = TextBold(B)
        ArtWork.FontItalic = TextItalic(B)
        wFlag = FlagAlign(B)
        TA = SetTextAlign(ArtWork.hDC, wFlag)
        ArtWork.Print Storedtext(B)
        ArtWork.AutoRedraw = False
        ArtWork.ScaleLeft = SaveScaleLeft
W(B) = TL - OS: E(B) = TW - OS: N(B) = TT: S(B) = TH
Txt.Caption = ""
End If
Txt.Visible = False
Txt.FontBold = False
Txt.FontItalic = False
Txt.Width = 15
        ArtWork.FontBold = False
        ArtWork.FontItalic = False
End Sub
```

下面一步一步地解释 ReDrawText 过程：

1. 在屏幕上输入的任何文字都放在标题控制对象 Txt 内。
2. 由于 Txt.Caption 是一个新对象，对象总数增 1，并将该值指定给此文字对象。
3. 指定文字的新边界方框坐标位置，其位置与文字标签相同。
4. 将 Txt 标题的标题文字存在数组 StoredText 中，将 Visual Basic 文字属性存在数组 TextSize，TextFont，TextBold 和 TextItalic 中。
5. 用下列方法在桌面上重画该文字：

    - 用 Print 语句放置文字，Print 语句需要知道用来放置文字的文字光标 CurrentX（当前 X 坐标）和 CurrentY（当前 Y 坐标）。你正在使用的文字光标是一个宽度被设为 1 个象素的标签控制对象。
    - CurrentX 等于该标题的 Left 属性值，用变量 TL 代表。CurrentY 等于标题的 Top 属性值，用 TT 变量表示。
    - **SaveScaleLeft**：文字串中出现一个回车符将使 CurrentX 设为 0，所以要保存 Artwork 图片框控制对象的当前 ScaleLeft 属性值。
    - **ArtWork.ScaleLeft = -ArtWork.CurrentX**：重设（复位）Artwork 的 ScaleLeft 属性，这样下一行就贴在第一行下面放置。当使用缩放工具时还要使用另一个名为 Scaling 的值。
    - 指定要绘制文字的所有属性。
    - **TextSize(B)**：代表要绘制文字点的大小，当正在使用缩放工具时，必须改变该大小。在本程序中，使用了一种简化的方法来解决此问题，也就是将点大小除以变量 TextPoint，TextPoint 的值在 ZoomPoint 过程（清单 5.19）中计算得到。有关最精确的文字点数学转换公式，请参考第 15 章中对 ArtAPI 程序的介绍。当用 API 视窗函数模拟缩放功能时，如果文字是用 API 图形文字函数重画的，那么它的大

小会自动改变。有关 API 视窗缩放的例子请参考第 14 章中的 Scaletest 程序。

- **WFlag=FlagAlign(B)**：用 API 的对齐函数设定左对齐、居中对齐或右对齐。有关 API 文字对齐的例子，请参见第 12 章中的 TxtAlign 程序。
- **ArtWork. Paint StoredText(B)**：打印(显示)存储的文字。
- 重设(复位)Artwork 的初始 ScaleLeft 属性，并保存文字的边界方框坐标。

5.8　重新设定节点和图柄的内部坐标比例

Rescale 过程用来在放大或缩小后更新全部节点和图柄的内部比例。Zoom 的值在 ZoomBar 事件中计算得到。

清单 5.8　缩放后，重新设定图片载体内控制对象的比例

```
Sub ReScale ()
BezHandle1. ScaleWidth = BezHandle1. Width + Zoom
BezHandle1. ScaleHeight = BezHandle1. Height + Zoom
BezHandle2. ScaleWidth = BezHandle2. Width + Zoom
BezHandle2. ScaleHeight = BezHandle2. Height + Zoom
BezNode1. ScaleWidth = BezNode1. Width + Zoom
BezNode1. ScaleHeight = BezNode1. Height + Zoom
BezNode2. ScaleWidth = BezNode2. Width + Zoom
BezNode2. ScaleHeight = BezNode2. Height + Zoom
''''''

For R = 0 To 3
    Node(R). ScaleLeft = Node(R). Left
    Node(R). ScaleTop = Node(R). Top
    Node(R). ScaleWidth = Node(R). Width + Zoom
    Node(R). ScaleHeight = Node(R). Height + Zoom
Next R
For R = 0 To 7
    Handle(R). ScaleLeft = Handle(R). Left
    Handle(R). ScaleTop = Handle(R). Top
    Handle(R). ScaleWidth = Handle(R). Width + Zoom
    Handle(R). ScaleHeight = Handle(R). Height + Zoom
Next R
Zoom = 0
End Sub
```

- **BezHandle1. ScaleWidth=**：重新设定两个 BezHandle 图片框控制对象的全部内部 Scale(比例)属性。

  BezNode1. ScaleWidth=：重新设定两个 BezNode 图片框控制对象的全部内部 Scale(比例)属性。

  Node(R). ScaleLeft=：重新设定两个 Node 图片框控制对象的全部内部 Scale(比例)属性。

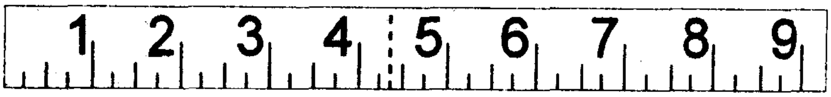
  Handle(R). ScaleWidth=：重新设定两个 Handle 图片框控制对象的全部内部 Scale(比例)属性。

5.9　标尺的鼠标器事件

两个标尺都是图片框控制对象，它们按英寸单位显示标尺刻度。标尺刻度是从桌面上所显示的页面的左上角 0,0 刻度处开始垂直、水平设置的。每个标尺还要显示一条点线标线，以指示鼠标指针在屏幕上的当前 X，Y 坐标。

清单 5.9　Ruler MouseDown 设置标尺新标线

```
Sub Ruler_MouseDown (Index%, Button%, Shift%, X!, Y!)
If BezHandle1.Visible = True Then ShowDotted = False: BezUpdate
    NewGuideLine = True
    X2 = -5000: Y2 = -5000
If Index = 0 Then SideRuler = True: Exit Sub
If Index = 1 Then TopRuler = True: Exit Sub
End Sub
```



If BezHandle1. Visible＝True：当准备拖动任一标尺上的标线时，如果选定了一条曲线，则在继续以后的操作之前必须更新该曲线。

NewGuideLine＝True：将该变量标志设为 True(真)表示正在将一条新标线拖至桌面。如果该标志被设为 False(假)，则只是将桌面上已经存在的标线拖至新位置。

X2＝－5000：Y2＝－5000：因为画标线用的是 twip 比例，所以必须使所画的每条线都足够长，能够穿过桌面区域。

If Index＝0 Then SideRuler＝True：侧边的标尺索引号为 0(Ruler(0))。如果正在拖动该标尺上的标线，则将 SideRuler 变量标志设为 True(真)。SideRuler 变量就会告诉 GuideLines 过程(清单 4.8)画一条垂直标线。

If Index＝1 Then TopRuler＝True：顶部标尺索引为 1(Ruler(1))。如果正在拖动该标尺上的标线，则将 TopRuler 变量标志设为 True(真)，TopRuler 变量就会告诉 GuideLines 过程(清单 4.8)画一条水平标线。

清单 5.10　画标线

```
Sub Ruler_MouseMove (Index%, Button%, Shift%, X!, Y!)
    Xstr = Format$ (X \ PixelX, "000")
    Ystr = Format$ (Y \ PixelY, "000")
XYLabel.Caption = "[" + Xstr + " , " + Ystr + "]"
        RulerGuides Index, Button, Shift, X, Y
    If Button Then GuideLines Index, Button, X, Y
End Sub
```

Xstr＝Format$：标尺是 Artwork 桌面上独立的图片框。所以，将鼠标的 X，Y 坐标传给

XYLabel 控制对象，以显示出鼠标指针的当前 X，Y 坐标。

RulerGuides 过程在每个标尺内画出移动的点线。如果按下鼠标键并保持住不放，则 GuideLines 过程就随着鼠标的拖动在 Artwork 桌面上画出点线标线。

清单 5.11 计算标线的新坐标

```
Sub Ruler_MouseUp (Index%, Button%, Shift%, X!, Y!)
X2 = X: Y2 = Y
    If NewGuideLine = True Then TGL = TGL + 1: V = TGL
    If SideRuler = True Then GuideX(V) = X2
    If TopRuler = True Then GuideY(V) = Y2
NewGuideLine = False
ArtWork.DrawStyle = Solid
        UpdateObjects
SideRuler = False: TopRuler = False: ShowDotted = False
End Sub
```

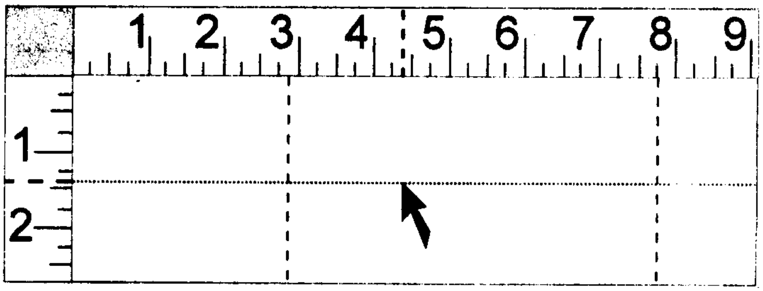


图 5.4 从标尺上拖出的标线

If NewGuideLine：Ruler_MouseDown 过程将 NewGuideLine 变量标记为 True，所以页面上标线的总数（TGL）增 1。

变量 V 代表指定给该标线的索引号。这与 BoundingBox_Up 过程（清单 2.20）中一样（当绘制一个新对象时，Total=Total+1：B=Total）。

GuideX(编号)=X2：存储垂直标线的 X 坐标，这样当要将该标线移到另一位置时，就可以找到它。

GuideY(编号)=Y2：存储水平标线的 Y 坐标，这样当要将该标线移到另一位置时，就可以找到它。

ArtWork.DrawStyle：将画图式样（DrawStyle）改为实线（Solid）。

UpdateObjects：重画桌面上所有的对象，然后将 SideRuler 和 TopRuler 标志复位为关闭（假）。

5.10 绘制每个标尺的滑线

RulerGuides 过程在每个标尺内绘制移动的点线。标尺滑线跟踪鼠标的当前 X，Y 坐标。

清单 5.12　绘制标尺滑线

```
Sub RulerGuides (Index%,Button As Integer, Shift As Integer,X!,Y!)
R0SL = Ruler(0).ScaleLeft 'Side Ruler
R0L = Ruler(0).Left-5000:R0T=Ruler(0).Top-5000
    R1ST=Ruler(1).ScaleTop 'Top Ruler
    R1L=Ruler(1).ScaleLeft-15:R1T=Ruler(1).ScaleTop-15
Ruler(0).Line(R0L,R0T)-(X1Ruler,Y1Ruler),Cyan,B 'Vert.
Ruler(1).Line(R1L,R0T)-(X2Ruler,Y2Ruler),Cyan,B 'Horz.
  X1Ruler=R0SL+500 'Side Ruler.Width
    Y1Ruler=Y
    X2Ruler=X
  Y2Ruler=R1ST+500 'Top Ruler.Height
Ruler(0).Line(R0L,R0T)-(X1Ruler,Y1Ruler),Cyan,B 'Vert.
Ruler(1).Line(R1L,R0T)-(X2Ruler,Y2Ruler),Cyan,B 'Horz.
End Sub
```

R0SL:(标尺 0 的 ScaleLeft)

R0L,R0T:(标尺 0 的 ScaleLeft,标尺 0 的 ScaleTop)当水平地或垂直地滚动桌面时,该标尺的 ScaleLeft 和 ScaleTop 可能会改变,所以用该标尺的当前内部比例开始绘制标尺滑线。

R1SL:(标尺 1 的 ScaleLeft)

R1L,R1T:(标尺 1 的 ScaleLeft,标尺 1 的 ScaleTop)当水平地或垂直地滚动桌面时,该标尺的 ScaleLeft 和 ScaleTop 可能会改变,所以用该标尺的当前内部 Scale(比例)开始绘制标尺滑线。

Ruler(0).Line;Ruler(1).Line:点线标尺线实际上是一个方框,该方框的三条边在视图区域以外,所以,你所看到的那条边看上去就像是一根线。给标准 X1,Y1 值指定一个负值,你就看不到该点线方框最外面的边。

5.11　在每个标尺内绘制英寸刻度

Ruler Paint 事件是在图片框或窗体内设置标尺刻度及数字的简洁明了、简便易行的方法。当你理解了这些 Line 语句是怎样递归地使用标尺的比例之后,就可以改变其度量比例和外观了。这里所用的度量值都是近似值,所以必须用计算器精确地设置 WYSIWYG(所见即所得)的度量单位。

清单 5.13　Ruler Paint 在每个标尺内画出英寸刻度

```
Sub Ruler_Paint (Index As Integer)
Ruler(0).Cls: Ruler(1).Cls
Ruler(0).DrawMode = 13: Ruler(1).DrawMode = 13
Ruler(0).DrawStyle = Solid: Ruler(1).DrawStyle = Solid
''''''''''''''''''''''''''''''''''

Ruler(0).ScaleLeft = 0: Ruler(0).ScaleWidth = 4: Num = -8
    For I = 300 To 15000 Step 576
```

```
        Ruler(0).Line (0, I)-(4, I)
        Ruler(0).Line (2, I - 288)-(4, I - 288)
        Ruler(0).Line (3, I - 144)-(4, I - 144)
        Ruler(0).Line (3, I - 432)-(4, I - 432)
        Ruler(0).PSet (-.5, I), Ruler(0).BackColor
        Num = Num + 1: Ruler(0).Print Num
    Next I
''''''''''''''''''''''''''''''''''
Ruler(1).ScaleTop = 0: Ruler(1).ScaleHeight = 4: Num = -12
    For I = 200 To 18000 Step 576
        Ruler(1).Line (I, 0)-(I, 4)
        Ruler(1).Line (I - 288, 0)-(I - 288, 2)
        Ruler(1).Line (I - 144, 0)-(I - 144, 1)
        Ruler(1).Line (I - 432, 0)-(I - 432, 1)
        Ruler(1).PSet (I, 1), Ruler(1).BackColor
        Num = Num + 1: Ruler(1).Print Num
    Next I
''''''''''''''''''''''''''''''''''
        Ruler(0).ScaleTop = ArtWork.ScaleTop
        Ruler(0).ScaleHeight = ArtWork.ScaleHeight
        Ruler(0).ScaleLeft = ScaleBox.Left + BSM
        Ruler(0).ScaleWidth = OldRW * PerSW
        Ruler(1).ScaleLeft = ArtWork.ScaleLeft
        Ruler(1).ScaleWidth = ArtWork.ScaleWidth
        Ruler(1).ScaleTop = ScaleBox.Top + SSM
        Ruler(1).ScaleHeight = OldRH * PerSH
''''''''''''''''''''''''''''''''''''
Ruler(0).DrawMode = Xor_Pen: Ruler(1).DrawMode = Xor_Pen
Ruler(0).DrawStyle = Dot: Ruler(1).DrawStyle = Dot
    XRuler = -5000: YRuler = -5000
    ArtWork.Enabled = True
End Sub
```

Ruler(0).DrawMode:标尺使用 Xor 绘图模式绘制点线标尺线，这些标尺线跟踪鼠标的 X，Y 位置。临时将它改为正常的默认值 Copy_Pen，以便画出英寸刻度。

Ruler.DrawStyle:临时将绘图式样改为默认值，实线(solid)。

Ruler(0).Scale:侧边标尺 ruler(0)的高度与 Artwork 桌面一样。顶部标尺 ruler(1)的高度与 Artwork 桌面相同。所需做的只是确定其内部比例以容纳用 Line 语句画出的每一个四分之一英寸刻度。

Num=-8:改变此数以调整英寸数字与下一个或上一个英寸刻度之间的距离。Num 值越高，英寸数在标尺中的位置就越高。

For I=300 To 15000 Step 576:从 300 开始重复计数，间隔为 576，直到 15000 个 twip 单位为止。这些值确定标尺上四分之一英寸、二分之一英寸、四分之三英寸及一英寸刻度的位置。你可以改变 Step(步长)值，精确地调整所有刻度的位置。如果改变起始值(300)和终止值(15000)，就可以调整标尺上的刻度个数，不过，你还必须精确调整变量 Num 和 Step(步长)的值。第 14 章中的 Scaletest 程序使用了同一个 Ruler_Paint 过程，

所以推荐你用这些值试试 Scaletest 程序。

下面是一些有关 Line 语句中 X1，Y1 和 X2，Y2 取值的例子：

Ruler(0). Line(0,I)－(4,I)：代表标尺上每一个 1 英寸刻度的位置。

Ruler(0). Line(2,I－288)－(4,I－288)：代表标尺上每一个 3/4 英寸刻度的位置。

Ruler(0). Line(3,I－144)－(4,I－144)：代表标尺上每一个 1/2 英寸刻度的位置。

Ruler(0). Line(3,I－432)－(4,I－432)：代表标尺上每一个 1/4 英寸刻度的位置。

数字 432，288，144 是标尺上每个刻度之间的 twip 距离。若要调整这些值，则请使用可相互整除的数，这样即可保持标尺刻度之间的距离相等。

Ruler(0). Pset：设定 X 值，向左或向右移动标尺数字，同时设定 Y 值，向上或向下移动标尺数字。

Ruler(0). Print I：显示标尺的英寸数。每循环一次，Num 计数就增 1。

若要设定恰当的、与 Artwork 相符的标尺对齐比例，请执行下列操作：

Ruler(n). Scale：重设两个标尺的比例，以反映 Artwork 的比例。变量 BSM，SSM，oldRW，oldRH，PerSW 和 PerSH 是在使用缩放工具或滚动桌面时应该加上的偏移值。这些值可以在 HScroll_Change 或 VScroll_Change 事件（清单 4. 14）中找到。

Ruler. DrawMode，DrawStyle，XRuler，YRuler：将标尺的属性全部设回 Ruler_Paint 事件开始前的值。

XRuler：YRuler：复位标尺标线的 X1，Y1 值。

ArtWork. Enabled＝True：启动时，Artwork 被取消对事件的反应，这样它就不会与 Ruler_Paint 事件发生冲突。

当用 Line 语句绘制标尺刻度时，必须加上每个刻度之间的间隔值。这些偏移值的位置取决于标尺是垂直的还是水平的。下面是一个例子：

```
Line(X1,Y1)－(X2,Y2)
```

垂直标尺

X1＝每个标尺刻度的长度

Y1＝每个标尺刻度之间的垂直距离

X2＝标尺的右边，也就是刻度线与标尺边线相交的地方

Y2＝与 Y1 相同

水平标尺

X1＝每个标尺刻度之间的水平距离

Y1＝标尺的顶边，也就是刻度线与标尺边线相交的地方

X2＝与 X1 相同

Y2＝每个标尺刻度的长度

5. 12　将值写进 ARTISAN. INI 文件

启动时，在 ColorBar 调色板中显示颜色所用的值是从 ARTISAN7. BAS 文件的默认列

表中取出的。这些颜色值位于子标题“Default Color Palette Value”下面三行。

清单 5.14 抽取数组 Colors，然后将它们写进一个 .INI 文件中

```
Sub SaveColor_INI ()
For I = 0 To 2
    ColorString = ""
        For C = 0 To 15
            ColorString = ColorString + [IC;CCC]
            Format $ (Colors(I * 16 + C), "00000000 ")
        Next C
    X = WritePrivateProfileString(KEY_NAME, KEY_ENTRY + [IC;CCC]
    Format $ (I + 1), ColorString, INI_FILENAME)
Next I
End Sub
```

以下几条发生在颜色调色板被装入之前：

1. Artisan 的 Form_Load 事件(清单 2.7)调用 Color_INI 过程(清单 4.5)。
2. Color_INI 过程使用一个 API 函数搜索一个自定义 .INI 文件。如果未找到，Color_INI 就使用位于 ARTISAN7.BAS 文件中的默认值。该 API 函数定义如下：

    ```
    GetPrivateProfileString(KEY_NAME,KEY_ENTRY+"1",BAS_PALETTE1,
        ColorString,Len(ColorString),INI_FILENAME)
    ```

    关键字名字在 ARTISAN7.BAS 文件中定义如下：

    ```
    Global Const KEY_NAME="Artisan"
    Global Const KEY_ENTRY="Colors Row"
    Global Const INI_FILENAME="Artisan.INI"
    ```

    默认的颜色调色板值在 ARTISAN7.BAS 中的定义如下所示：

    ```
    Global Const BAS_PALETTE1="16777215 14737632 12632319... 等等
    Global Const BAS_PALETTE2="12632256 04210752 08421631... 等等
    Global Const BAS_PALETTE3="08421504 00000000 00000255... 等等
    ```

3. PaletteBarColors 过程(清单 5.5)被调用三次，以抽取存在缓冲串 ColorString 中的值。值 0、16 和 32 是几个参数，这些参数用来改变计数每种颜色的变量 FirstColor 的值。
4. Colors(C)数组存放每种颜色的索引号，范围从 0 至 47。
5. 最后，ColorBar_Paint 事件(清单 4.3)调用 Display_Color_Palette(附录 A)，将该图片框划分成 48 个段，并指定每一个调色板项。

5.13 将某个对象放到所有其他对象的前面或后面

To_Front_Back 过程交换每个对象的值，以便把一个对象移到桌面上所有其他对象的上面或下面。该过程很长，它只适用于在项目中处理少量的对象。第 15 章的 ArtAPI 程序中

有一个短得多的 To_Front_Back 过程。ArtAPI 方法将所需的大量代码减缩成只有几行。不过嵌套的循环可能会使初学者迷惑，而且还必须使用额外的数组才能按正确的顺序重画所有对象。

清单 5.15　将一个对象送至所有其他对象的前面或后面

```
Sub To_Front_Back (Index As Integer)
Dim PC As Long: Dim OLC As Long
Dim St As String: Dim TF As String
If Node(0).Visible = False Then Exit Sub
If Index = 0 And B = Total Then Exit Sub
If Index = 1 And B <= 1 Then Exit Sub
''''''store objects attributes
If FlagText(B) = B Then FT = 1
If FlagCircle(B) = B Then FC = 1
If FlagLine(B) = B Then FL = 1
If FlagBox(B) = B Then FB = 1
If FlagBezier(B) = B Then Bez = 1
OLC = OutLineColor(B): PC = PaintColor(B): LW = LineWidth(B)
Fill = FillValue(B): LV = LineValue(B): St = Storedtext(B)
TS = TextSize(B): TF = TextFont(B): TB = TextBold(B): TI = TextItalic(B)
FA = FlagAlign(B): OF = OFFSET(B): CR = CarriageCount(B)
W1 = W(B): N2 = N(B): E3 = E(B): S4 = S(B)
p1XX = p1X(B): p1YY = p1Y(B): p2XX = p2X(B): p2YY = p2Y(B)
h1XX = h1X(B): h1YY = h1Y(B): h2XX = h2X(B): h2YY = h2Y(B)
'''''''End of every possible attribute
If Index = 0 Then Last = Total - 1: First = 1 Else Last = 2: First = -1
For B = B To Last Step First: A = B + (First)
W(B) = W(A): N(B) = N(A): E(B) = E(A): S(B) = S(A)
p1X(B) = p1X(A): p1Y(B) = p1Y(A)
P2X(B) = p2X(A): p2Y(B) = p2Y(A)
h1X(B) = h1X(A): h1Y(B) = h1Y(A)
h2X(B) = h2X(A): h2Y(B) = h2Y(A)
OutLineColor(B) = OutLineColor(A)
PaintColor(B) = PaintColor(A)
LineWidth(B) = LineWidth(A)
FillValue(B) = FillValue(A)
LineValue(B) = LineValue(A)
Storedtext(B) = Storedtext(A)
TextSize(B) = TextSize(A)
TextFont(B) = TextFont(A)
TextBold(B) = TextBold(A)
TextItalic(B) = TextItalic(A)
FlagAlign(B) = FlagAlign(A)
OFFSET(B) = OFFSET(A)
CarriageCount(B) = CarriageCount(A)
If FlagText(A) = 0 Then FlagText(B) = 0 Else FlagText(B) = B
If FlagCircle(A) = 0 Then FlagCircle(B) = 0 Else FlagCircle(B) = B
If FlagLine(A) = 0 Then FlagLine(B) = 0 Else FlagLine(B) = B
If FlagBox(A) = 0 Then FlagBox(B) = 0 Else FlagBox(B) = B
If FlagBézier(A) = 0 Then FlagBézier(B) = 0 Else FlagBézier(B) = B
Next B

''''Reset objects attributes
If FT = 1 Then FlagText(B) = B Else FlagText(B) = 0
If FB = 1 Then FlagBox(B) = B Else FlagBox(B) = 0
If FC = 1 Then FlagCircle(B) = B Else FlagCircle(B) = 0
If FL = 1 Then FlagLine(B) = B Else FlagLine(B) = 0
If Bez = 1 Then FlagBézier(B) = B Else FlagBézier(B) = 0
OutLineColor(B) = OLC: PaintColor(B) = PC: LineWidth(B) = LW
FillValue(B) = Fill: LineValue(B) = LV: Storedtext(B) = St
TextSize(B) = TS: TextFont(B) = TF: TextItalic(B) = TB: TextItalic(B) = TI
FlagAlign(B) = FA: OFFSET(B) = OF
CarriageCount(B) = CR
W(B) = W1: N(B) = N2: E(B) = E3: S(B) = S4
p1X(B) = p1XX: p1Y(B) = p1YY: p2X(B) = p2XX: p2Y(B) = p2YY
h1X(B) = h1XX: h1Y(B) = h1YY: h2X(B) = h2XX: h2Y(B) = h2YY
UpdateObjects
End Sub
```

If Node(0)：没有选定任何对象，所以退出此过程。

If Index=0 And B=Total：你已经选择了将该对象移至前面，但是该对象已经在前面，所以退出此过程。

If Index=1 And B<=1：你已经选择了将该对象移至后面，但是该对象已经在后面，所以退出此过程。

If FlagText(B)=B Then FT=1：查出要移动的对象的类型(圆，方框，直线)。如果该对象类型的标志等于B(对象的指定编号)，则找到了。

OLC=OutlineColor(B)：PC=PaintColor(B)，等等：临时保存该对象可能已指定的全部属性。

W1=W(B)，N2=N(B)，E3=E(B)，S4=S(B)：临时存储该对象的当前边界方框坐标。

下一步是确定排序方向——升序(送至前面)还是降序(送至后面)。

If Index=0 Then Last=Total−1：First=1 Else Last=2：First=−1：如果选定的菜单索引号等于0，则将对象移至前面，变量Last就等于对象总数减1，并且变量First等于1。

Else：否则将对象移至后面，变量Last就等于2，变量First等于−1。给定一个负的step(步长)值(例如−1)就会使得For/Next语句反向计数，于是该对象就移至后面而不是前面。

For/Next：执行排序循环。

For B=B To Last Step First：A=B+(First)：从当前对象(B)开始，直到最后一个对象为止，对于每一个对象，执行下面的操作。For/Next语句内列出的每一个属性数组的值都将被移至数组中下一次属性索引号。B值代表当前对象的指定索引号，A值代表下一个索引号B+1(升序)或B−1(降序)。

If FlagText(A)＝0 Then FlagText(B)＝0 Else FlagText(B)＝B：这将确保每个对象都被赋予一个正确的对象标志名。UpdateObjects 过程必须知道该标志名才能重画正确的对象。

Next B：对下一个对象重复执行。

除了正要移动的对象外其他的对象都被更新。该过程的最后部分就是将选定的对象放在所有其他对象的前面或后面。

W(B),B(B),E(B),S(B)：恢复对象初始边界方框的位置。

p1X(B),p1Y(B),等等：如果该对象是一条曲线，则恢复该曲线的 4 个控制点。

OutlineColor(B),PaintColor(B),等等：恢复该对象的全部初始属性。

If T＝1：该对象是文字，所以恢复此标志。**Else**：否则关闭该标志。

If FT＝1 Then FlagText(B)＝B Else FlagText(B)＝0：在本过程开始时检查了选定移动的是哪一种对象，现在恢复该对象的初始标志名。

UpdateObjects：按适当的顺序重画所有对象。

5.14　重画桌面上的所有对象

桌面上的所有对象，以及页面的隐含图象都在 UpdateObjects 过程中重画。每当创建一个对象时，在 BoundingBox_Up 过程(清单 2.20)中，该对象就被指定了一个标志名和一个对象编号。该对象的指定编号与代表该标志名(例如 FlagBox(B),FlagCircle(B))的数组内的索引项相匹配。例如，如果在桌面创建的第 3 个对象是一个圆，则 FlagCircle 数组的索引 3 中就含有值 3(FlagCircle(3))，所有其他标志名数组的索引 3 位置的值都为 0。

本程序用了一个 DO/LOOP 语句来遍历桌面上的所有对象(B)。然后，就可以用一个 Select/Case 语句来查找与标志名索引值相匹配的对象号(B)。第 15 章的 ArtAPI 程序中，一个类似的 UpdateObjects 过程使用了 For/Next 语句来遍历桌面上的所有对象而没有用 DO/LOOP 语句。

清单 5.16　重画隐含页面及其上面的所有对象

```
Sub UpdateObjects ()
LastMove = B
ArtWork.DrawMode = Copy_Pen
ArtWork.DrawStyle = Solid
ArtWork.AutoRedraw = True
If Draw = True Then GoTo SkipErase
ArtWork.DrawWidth = 20
ArtWork.Line (W(B), N(B))-(E(B), S(B)), White, BF
ArtWork.Line (XX2, YY2)-(XX1, YY1), White, BF: B = 1
ArtWork.DrawWidth = 1
        UpdateDeskTop
Do While B < Total + 1
    ArtWork.DrawWidth = LineWidth(B)
    If LineValue(B) > 0 Then ArtWork.DrawStyle = 5 Else [IC:CCC]
    ArtWork.DrawStyle = Solid
SkipErase:
```

```
Select Case B
    Case FlagCircle(B)
        X1 = W(B): Y1 = N(B): X2 = E(B): Y2 = S(B)
        ArtWork.DrawWidth=LineWidth(B)
        GoodCircle = PerfectCircle(X1, Y1, X2, Y2)
        ArtWork.FillColor = PaintColor(B):
        ArtWork.FillStyle = FillValue(B)
        ArtWork.Circle (XCenter, YCenter), Radius, [IC:CCC]
        OutLineColor(B), , , Aspect
        ArtWork.FillColor = 0: ArtWork.FillStyle = 1
    Case FlagLine(B)
      ArtWork.Line (W(B), N(B))-(E(B), S(B)), OutLineColor(B)
    Case FlagBox(B)
      If FillValue(B) = 1 Then GoTo SkipFill
      ArtWork.Line (W(B), N(B))-(E(B), S(B)), PaintColor(B), BF
SkipFill:
      ArtWork.Line (W(B), N(B))-(E(B), S(B)), OutLineColor(B), B
  Case FlagBézier(B)
    p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
    h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
    ArtWork.ForeColor = OutLineColor(B)
    BezDraw
  Case FlagText(B)
    ArtWork.CurrentX = OFFSET(B): ArtWork.CurrentY = N(B)
    SaveScaleLeft = ArtWork.ScaleLeft
    ArtWork.ScaleLeft = -ArtWork.CurrentX + Scaling
    ArtWork.FontName = TextFont(B)
    ArtWork.ForeColor = PaintColor(B)
    ArtWork.FontSize = TextSize(B) * TextPoint
    ArtWork.FontBold = TextBold(B)
    ArtWork.FontItalic = TextItalic(B)
    wFlag = FlagAlign(B)
    TA = SetTextAlign(ArtWork.hDC, wFlag)
    ArtWork.Print Storedtext(B)
    ArtWork.ScaleLeft = SaveScaleLeft
End Select
If Draw = True Then Draw = False:[IC:CCC]
    ArtWork.AutoRedraw = False: B = LastMove: Exit Sub
  B = B + 1
Loop
  B = LastMove
    ArtWork.AutoRedraw = False
    ArtWork.DrawWidth = 1:
    ArtWork.DrawStyle = Solid
If DeleteObject = True Then Exit Sub
If CurrentTool = 1 Or CurrentTool = 2 Then Exit Sub
If SideRuler = True Or TopRuler = True Then Exit Sub
    Handles
    Nodes
End Sub
```

LastMove＝B：将当前对象的指定编号临时存储在变量LastMove中。如果对象是因为使用了To_Front_Back过程(清单5.15)而被重画的，则在所有其他对象被重画后，用节点和图柄高亮显示被移动的对象。变量LastMove在本过程末尾用来将节点和图柄指定给最后选定的对象。

ArtWork.DrawMode：用默认的Copy_Pen绘图模式重画对象。

ArtWork.DrawMode：用默认的solid(实线)绘图式样重画对象。

If Draw＝True Then GoTo SkipErase：当Draw标志被设为真(true)时，则说明你刚刚画了或拉伸了一个位于所有其他对象上面的对象，因此没有必要重画位于该对象下面的所有对象(该对象位于最上面)。可以跳过DO/LOOP语句，转至SkipErase，仅重画最上面的对象。

下面三行用来擦除上一个位置处的对象边界方框。

ArtWork.DrawWidth：将边界方框边缘宽度设为20个象素，这样就可以保证任何边框很宽的对象都可以擦掉其边框。

ArtWork.Line(W(B),N(B))－(E(B),S(B))：当前对象边界方框的大小。该区域被桌面颜色(白色)填充擦掉。

ArtWork.Line(XX1,YY1)－(XX2,YY2)：如果该对象被移至新位置或被拉伸，则用桌面颜色(白色)填充该方框以擦掉原有的图象。

UpdatePageSize：置中在桌面上的白色页面是通过画两个方框构造出来的。第一个方框用黑色填充，另一个方框则用白色填充，并稍稍向左移动。这将产生一种桌面上有一张纸的视觉效果。页面的绘制是在UpdatePageSize过程(清单5.17)中完成的。现在就可以在Artwork图片框控制对象(桌面)上重画对象了。

Do While B<Total＋1：循环遍历桌面上的全部对象。执行循环直至对象编号不小于所有对象的总数加1为止。

ArtWork.DrawWidth＝LineWidth(B)：设置每一个要画的对象的宽度。

If LineValue(B) ＝ 0 Then：如果一个对象的LineValue变量等于0，则说明该对象没有边框。不能为了不画出某个对象的边框而将Artwork的DrawWidth设为0。对于每个没有边框的对象，必须将Artwork的绘图式样(DrawStyle)设为不可见(5)。

SkipErase：如果Draw标志被设为真(true)，则跳过Do/Loop语句。

Select Case B：B代表对象的指定编号。

Case FlagCircle(B)：该对象是一个圆，所以该圆的边界方框大小X1,Y1,X2,Y2被传递给PerfectCircle函数(清单5.6)以计算出圆的半径和中心位置。

ArtWork.Circle：用PerfectCircle过程返回的值画圆。

Case FlagLine(B)：该对象是一条直线，所以，简单地使用Line语句重画它。

Case FlagBox(B)：该对象是一个方框，所以用Line语句的Box(方框)选项重画它。如果该对象的FillValue变量等于0，则画出一个实心方框。如果FillValue变量等于1，则转至SkipFill代码行，画一个非填充方框。

Case FlagBézier：该对象是一条Bézier曲线，因此取得该曲线的4个控制点。BezDraw过程(清单3.1)将重画该曲线。

Case FlagText：该对象是文字，所以重画该文字串。关于怎样在本程序中重画文字，请参考 ReDrawText 过程（清单 5.7）。

ArtWork. CurrentX：你正在用 Visual Basic 的 Print 语句绘制文字，所以用 CurrentX，CurrentY 语句设置文字的起始点。CurrentX 被 ReDrawText 过程存在数组 OffSet 中，CurrentY 是文字边界方框的北边（顶边）坐标。

ArtWork. ScaleLeft：一个文字串被一个内嵌的回车符折至下一行时，下一行文字的 CurrentX 或起始点就被设成 0。将 Artwork 的 ScaleLeft 属性临时改为第一行文字的 CurrentX 位置，并减去这些值，以对齐下面的文字。

ArtWork. FontName：文字对象所使用字体的名字。

ArtWork. ForeColor：字体的颜色。

ArtWork. FontSize：所用文字的点大小。

ArtWork. FontBold：这是一个确定该字体是否有 Bold（粗体）属性的标志。

ArtWork. FontItalic：这是一个确定该字体是否有 Italic（斜体）属性的标志。

wFlag：这是 Windows API 中用来设定左对齐、居中对齐或右对齐的标志。

TA=SetTextAlign：这是一个设定对齐属性的 API 函数。ArtWork. hDC 是正在使用的图片框的名字。wFlag 是 API 的文字对齐标志。

ArtWork. Print：打印（显示）存储在 StoredText(B)数组中的文字串。

ArtWork. Scale：将 Artwork 的 Scale（比例）复位为初始值。

If Draw=True Then：如果变量 Draw 被设成 True（真），则此过程仅重画一个对象，然后退出。关闭 AutoRedraw，并确保 LastMove 中存储的值被赋给了当前对象。

B=B+1：这使得对象编号增 1，这样，就可以重画下一个对象了。

Loop：对下一个要画的对象重复执行该 Do/Loop 语句块。

B=LastMove：循环最终重画了所有对象。如果在此过程执行之前，某个对象的边界方框随同图柄和节点一起被高亮显示，则告诉程序使其继续高亮显示。循环改变了 B 的值，但是在本过程的开始处就将原来的 B 变量值存放在 LastMove 变量中，使 B 重新等于当前对象编号。

ArtWork. AutoRedraw：关闭 AutoRedraw。

ArtWork. DrawWidth：将绘图宽度重新设回默认值 1。

ArtWork. DrawStyle：将绘图式样重新设回默认值 Solid（实线）。

If DeleteObject=True：删除一个对象，所以退出该过程，而不显示其图柄和节点。

If CurrentTool = the BézierTool：不要显示图柄和节点，因为 Bézier 曲线有自己的 BezHandles 和 BezNodes 过程，所以退出本过程。

If SideRuler=True Or TopRuler=True Then Exit Sub：该语句意味着所要更新的是以前绘制的标线。因此不必显示图柄或节点，退出本过程。

Handles：如果到现在还没有退出，则要显示对象的图柄。

Nodes：还要显示出对象周围的节点。

5.15 重画隐含页面

UpdateDeskTop 过程的第一部分绘制显示在 Artwork 桌面中央的页面上，UpdateDesk-

Top 过程的第二部分重画可能出现在桌面上的任何点线标线。标线是从两个标尺中拖出放在桌面上的。

清单 5.17　重画标线和隐含页面

```
Sub UpdateDeskTop ()
SW = ((ArtWork.Width) \ 2) * PixelX 'Redraw Page
SH = ((ArtWork.Height) \ 2) * PixelY
OS = 75
ArtWork.DrawWidth = 1
    PageX1 = SW - (163 * PixelX) * 1
    PageY1 = SH - (212 * PixelY) * 1
    PageX2 = SW + (162 * PixelX) * 1
    PageY2 = SH + (212 * PixelY) * 1
    ArtWork.Line (PageX1 + OS, PageY1 + OS)-[IC:CCC]
          (PageX2 + OS, PageY2 + OS), GRAY, BF
     ArtWork.Line (PageX1, PageY1)-(PageX2, PageY2), White, BF
     ArtWork.Line (PageX1, PageY1)-(PageX2, PageY2), Black, B
''''''''''''''''''

For R = 1 To TGL 'Redraw Any Guide Lines
     ArtWork.DrawStyle = Dot
     ArtWork.Line (-5000, -5000)-(GuideX(R), 24000), QBColor(1), B
     ArtWork.Line (-5000, -5000)-(24000, GuideY(R)), QBColor(1), B
Next R
    X2 = -5000: Y2 = -5000
ArtWork.DrawStyle = Solid
End Sub
```

SW：代表屏幕（监视器）宽度的中心点。

SH：代表屏幕（监视器）高度的中心点。

OS：代表用来产生页面视觉深度的偏移值。

ArtWork.DrawWidth：将要绘制的页面边界宽设置为一个象素。

PageX1＝SW－(163＊PixelX)＊1：等于页面的左边界。如果所用的监视器是一个大监视器，则可以将乘数＊1 改为＊2。

PageY1：等于页面的顶边。

PageX2：等于页面的右边界。

PageY2：等于页面的底边。

ArtWork.Line #1：根据上面的页面尺寸画一个实心的黑色方框，并加上偏移(OS)产生阴影效果。

ArtWork.Line #2：使用不带偏移的页面尺寸，在黑色方框的上面用白色再画一个实心方框。

ArtWork.Line #3：在白色方框周围画一条黑色边界线。

重画标线：

For R＝1 To TGL：计数器 R 从 1 开始计数，直到目前所画的标线总数为止。变量 TGL

保存到目前为止所画的标线的总数。

ArtWork. DrawStyle：将要重画标线，标线是点线，所以将绘图式样改为 Dot（点线）。

ArtWork. AutoRedraw：如果当前 AutoRedraw 是关闭的，则将其打开。

ArtWork. Line＃1：重画垂直标线。

ArtWork. Line＃2：重画水平标线。

Next R：重复处理下一组标线（如果还有）。

ArtWork. DrawStyle：将绘图式样改回 Solid（实线）。

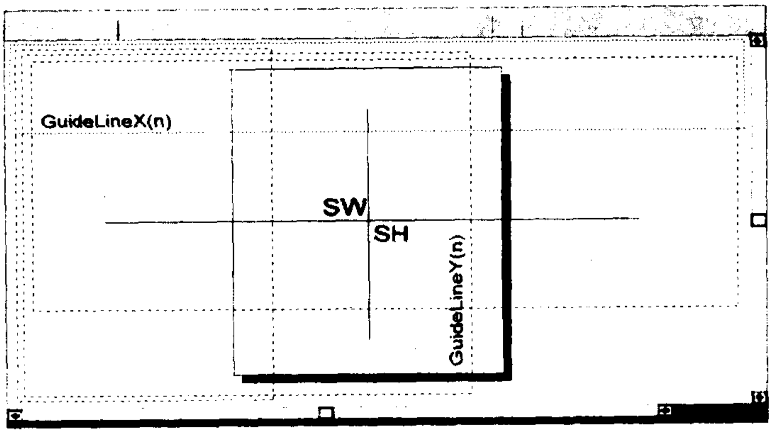


图 5.5　桌面页面的布局

5.16　向打印机传送图象

WinPrint 过程与 UpdateObjects 过程（清单 5.16）类似，不同的是在打印机页面上而不是在监视器上显示对象。例如不是在 Artwork 的桌面上绘制对象（ArtWork. Line(X1,Y1)－(X2,Y2)），而是改成下面的语句（(Printer. Line(X1,Y1)－(X2,Y2))。

清单 5.18　WinPrint 过程将对象传送给打印机

```
Sub WinPrint ()
    On Error GoTo PrintErr:
CMDialog1.CancelError = True
CMDialog1.Flags = &H2& Or &H4& Or &H8& Or &H20&
CMDialog1.Action = 5
NumCopies = CMDialog1.Copies
Screen.MousePointer = 11

    For r = 1 To NumCopies
    PrinterScale = Printer.ScaleWidth 'Save scale for fontsize
    Printer.ScaleLeft = PageX1 'Left Margin
    Printer.ScaleTop = PageY1     'Top Margin
    Printer.ScaleWidth = (PageX2 - PageX1) 'Right Enlarge
    Printer.ScaleHeight = (PageY2 - PageY1)'Bottom Enlarge
B = 1
Do While B < Total + 1
  Printer.DrawWidth = LineWidth(B)
  If LineValue(B) > 0 Then Printer.DrawStyle = 5 Else Printer.DrawStyle = Solid
```

```
   Select Case B
   Case FlagCircle(B)
          X1 = W(B): Y1 = N(B): X2 = E(B): Y2 = S(B)
          GoodCircle = PerfectCircle(X1, Y1, X2, Y2)
          Printer.FillColor = PaintColor(B): Printer.FillStyle = FillValue(B)
          Printer.Circle (XCenter, YCenter), Radius, OutLineColor(B), , , Aspect
          Printer.FillColor = 0: Printer.FillStyle = 1
   Case FlagLine(B)
      Printer.Line (W(B), N(B))-(E(B), S(B)), OutLineColor(B)
   Case FlagBox(B)
      If FillValue(B) = 1 Then GoTo SkipFill2
      Printer.Line (W(B), N(B))-(E(B), S(B)), PaintColor(B), BF
 SkipFill2:
      Printer.Line (W(B), N(B))-(E(B), S(B)), OutLineColor(B), B
    Case FlagBézier(B)
      p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
      h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
      Printer.ForeColor = OutLineColor(B): FlagPrinter = 2
      BezDraw
    Case FlagText(B)
      Printer.CurrentX = OFFSET(B): Printer.CurrentY = N(B)
      SaveScaleLeft = Printer.ScaleLeft
      Printer.ScaleLeft = -(OFFSET(B) - Printer.ScaleLeft)
      Printer.FontName = TextFont(B)
      PFontScale! = (PrinterScale) / (Printer.ScaleWidth)
      Printer.FontSize = TextSize(B) * PFontScale!
      Printer.FontBold = TextBold(B)
      Printer.FontItalic = TextItalic(B)
      Printer.ForeColor = PaintColor(B)
      wFlag = FlagAlign(B)
      TA = SetTextAlign(Printer.hDC, wFlag)
      Printer.Print Storedtext(B)
      Printer.ScaleLeft = SaveScaleLeft
End Select
'''''
B = B + 1
Loop
      Printer.EndDoc
Next r
      Screen.MousePointer = 0
      FlagPrinter = 0
      Exit Sub
PrintErr:
      Screen.MousePointer = 0
      FlagPrinter = 0
      Exit Sub
End Sub
```

CMDialog1：有关CMDialog控制对象的方法及属性值的解释，请参考《Visual Basic语言指南》中的“公共对话框控制对象”和“过滤筛选器”。

PrinterScale=Printer. ScaleWidth：保存打印机的当前比例。当打印每一行文字时，本过程的 Select/Case FlagText 段改变了打印机的 ScaleLeft。PrinterScale 变量在 Select/Case FlagText 段中被用来重新指定打印机所需的适当比例。

Printer. ScaleLeft=PageX1：左边空距值。改变该值调整或移动图象，以使其靠近或远离页面左边界。

Printer. ScaleTop=PageY1：顶部边距值。改变该值调整或移动图象，以使其靠近或远离页面顶边界。

Printer. ScaleWidth=(PageX2-PageX1)：页面宽度。改变该值，可垂直缩放图象。

Printer. ScaleHeight=(PageY2-PageY1)：页面高度。改变该值，可水平缩放图象。

Case FlagText(B)：在显示文字时，FlagText 块与 UpdateObjects 过程的方法稍有不同。Printer. ScaleLeft 语句现在必须反映出打印机的内部比例，而不是 Artwork 图片框控制对象的内部比例。字体大小必须翻倍才能反映出实际打印的点大小。手工获得此结果的最好方法是用 Windows 的 Write 程序在一页纸上打印出一些文字，然后将它与本程序打印出的文字相比较。第 15 章中的 ArtAPI 程序采用了 API 文字函数，绕过了手工确定文字点大小比例的方法。有关可用来计算文字点大小的数学公式请参考对 ArtAPI 程序的介绍。

5.17 使用缩放工具

ZoomTool 过程用简单的人工缩放来得到两种缩放视图状态(大或小)。API 的比例缩放函数比 Visual Basic 的比例缩放法更高出一筹，而且能够提供更大范围的比例和更高的精度。还请注意的是，这里所展示的放大(Zoom In)和缩小(Zoom Out)功能并没有使用精确的比例单位，而只是一些粗略的百分比。

附带磁盘 16MISC 目录下的 APIScale 程序给出了一个怎样在桌面选定区域进行缩放的例子，它还演示了怎样在 Visual Basic 版本 3.0 或更低版本中将鼠标指针改为一个自定义的缩放工具图标。

在附带磁盘的 14SCALE 目录下还可以找到另外两个使用手工及 API 比例缩放功能的程序(Rulers 和 Scaletest)。

清单 5.19 改变桌面比例，造成缩放效果

```
Sub ZoomBar_MouseDown (Button%, Shift%, X!, Y!)
CurrentTool = 2: ToolButtons_Paint:
CurrentTool = LastToolUsed: ToolButtons_Paint
CurrentZoom = X \ 32: ZoomBar.Visible = False: N_H_Clear
Select Case CurrentZoom
    Case ZoomIn 'First Button
        If Zoom_Out = True Or Zoom_In = True Then Beep: Exit Sub
        ArtWork.AutoRedraw = True: ArtWork.Cls: ArtWork.AutoRedraw = False
        Scaling = 3750: Zoom_In = True: Zoom_Out = False
        Zoom = -48: NodeZoom = - 21: TextPoint = 1.7
        Txt.FontSize = txt.FontSize * TextPoint
        ZoomScale 2, 3750, 3750, -7500, -6375
    Case ZoomOut 'Second Button
        If Zoom_Out = True Or Zoom_In = True Then Beep: Exit Sub
        ArtWork.AutoRedraw = True: ArtWork.Cls:
        ArtWork.AutoRedraw = False
        Scaling = -5000: Zoom_In = False: Zoom_Out = True
        Zoom = 65: NodeZoom = 28: TextPoint = .65
        Txt.FontSize = txt.FontSize * TextPoint
        ZoomScale 2, -5000, -5000, 10000, 8500
    Case ZoomAll 'Third Button
    X1 = W(1): Y1 = N(1): X2 = E(1): Y2 = S(1)
      For I = 1 To Total
        If X1 > W(I) Then X1 = W(I)
        If Y1 > N(I) Then Y1 = N(I)
        If X2 < E(I) Then X2 = E(I)
        If Y2 < S(I) Then Y2 = S(I)
        Next I
''''''
    For I = 1 To Total
        If X1 > E(I) Then X1 = E(I)
        If Y1 > S(I) Then Y1 = S(I)
        If X2 < W(I) Then X2 = W(I)
        If Y2 < N(I) Then Y2 = N(I)
    Next I
''''''
ArtWork.Line (X1, Y1)-(X2, Y2), QBColor(4), B
    Case ZoomPage 'Fourth Button
        ArtWork.AutoRedraw = True:
        ArtWork.Cls: ArtWork.AutoRedraw = False
        ArtWork.Scale: PerSW = 1: PerSH = 1
        Zoom = 0: NodeZoom = 0: TextPoint = 1
        Scaling = 0: Zoom_In = False: Zoom_Out = False
    Txt.FontSize = 12
        For R = 0 To 7
            Handle(R).Scale
        Next R
        For R= 0 To 3
            Node(R).Scale
        Next R
    ArtWork.MousePointer = 1
    VScroll.Value = 0: HScroll.Value = 0
    ZoomScale 2, 0, 0, 0, 0
End Select''''''''''''''''''''''''''''''
        StatusArea_Paint
ToolButtons_MouseDown Button, Shift, X, Y
End Sub
```

Select Case CurrentZoom：你可从四个可选的缩放按钮中选择任意一个。

Case ZoomIn：从左边数起的第 1 个按钮。用来在桌面做放大操作。

If ZoomOut：防止多步缩放。因为当 Artwork 的比例被改变时，所有节点和图柄都必须

重新设定比例(缩放)，所以，应该有一个特殊的函数计算每一个变化的百分比，这样，在缩放后就可以正确对齐节点和图柄。

ArtWork. AutoRedraw:Cls:从内存中删去桌面的当前位图图象。

现在，你就可以通过改变 Artwork 的比例来进行缩放了。将左边比例增加 Artwork 正常 ScaleWidth(15000)的四分之一，并将 Artwork 的正常 ScaleWidth 减半(7500)，就可以将缩放百分比增加到25%左右，并仍然将图象保持在屏幕(控制对象)的中央。

必须保持 Artwork 的 ScaleHeight 的正确纵横比。将顶边比例减少 ArtWork 正常 ScaleHeight(18000)的四分之一，并将其正常的 ScaleWidth 减少85%，就可以确定与宽度百分比相适应的高度比例(假设使用的是 SVGA 监视器)。

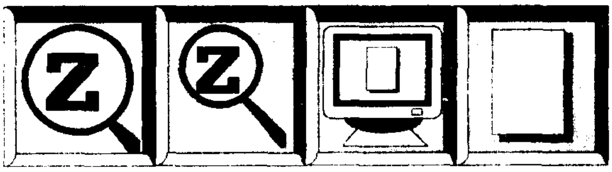
Scaling=3750:代表 Artwork 的新 ScaleLeft 值。改变比例位移也需要 Scaling 的值，比例位移被标尺用来反映新的比例大小。

Zoom:被选定对象周围的8个图柄(红色方框)也需要重新设定比例(缩放)。在缩放过程中，这些控制对象都不会增大或缩小，它们是位于绘制对象上独立的控制对象。例如，放大(Zoom In)操作会增加所绘图形的大小和比例，但图柄却仍保持原有大小和比例不变。所以，必须计算出 Artwork 的新比例和图柄控制对象内部比例之间差值的百分比。Zoom 变量就代表给每个图柄控制对象所加的偏移。

NodeZoom:给每个节点控制对象加的偏移，详情请参见 Zoom。

TextPoint=1.7:代表文字点大小增加的百分比，它反映出 Artwork 的新比例。请注意，值1.7是一个大约的百分比。Visual Basic 将点大小按照四分之一的小数值进行舍入(例如，8.25点)，用一个公式来取得精确的文字点大小不会给我们带来任何好处。

ZoomScale 2:将新比例大小传给 ZoomScale 过程(清单5.20)以进行重新缩放。



Case ZoomAll:这是从左数起的第3个缩放按钮，它缩放所有对象，以填满整个屏幕。它在本程序中并不起作用，而最好是同 API 比例函数一起使用。不过该过程展示了怎样找到至目前为止所画的所有对象，并在它们周围画一个方框，该方框即代表要放大的区域。一旦建立起了一种缩放方法后，就可将 X1、Y1、X2、Y2 的值用作新窗口的尺寸进行缩放。

Case ZoomPage:从左数起的第四个缩放按钮。该按钮复位至页面的默认视图。

ArtWork. AutoRedraw:用 Cls(清除屏幕)语句擦除 ArtWork 中包含的所有图象。

Zoom:NodeZoom:等等:将所有的值复位为正常值。

Handle(R):Node(R):将所有图柄和节点复位为正常比例。

VScroll,HScroll:将滚动条复位至中心位置。

ZoomScale 2:传递0值作参数，将比例复位为正常值。

ToolButtons MouseDown:给 ToolButtons(工具按钮)图片框发送一个单击事件，高亮显

示某个工具。

5.18　重新设定桌面视窗的比例

新的比例尺寸在 ZoomBar_MouseDown 事件(清单 5.19)的 Zoom_In，Zoom_Out 或 Zoom_Page 代码块中是作为参数(四个数字值)来传递的。

清单 5.20　改变桌面的比例

```
Sub ZoomScale (Index%, ZLeft%, ZTop%, ZWidth%, ZHeight%)
   ArtWork.ScaleLeft = ArtWork.ScaleLeft + ZLeft
   ArtWork.ScaleTop = ArtWork.ScaleTop + ZTop
   ArtWork.ScaleWidth = ArtWork.ScaleWidth + ZWidth
   ArtWork.ScaleHeight = ArtWork.ScaleHeight + ZHeight
   NewSW = ArtWork.ScaleWidth: NewSH = ArtWork.ScaleHeight
   PerSW = NewSW / OldSW: PerSH = NewSH / OldSH
   BSM = BSM * PerSW: SSM = SSM * PerSH
'''''''''''
           Ruler(0).ScaleTop = ArtWork.ScaleTop
           Ruler(0).ScaleHeight = ArtWork.ScaleHeight
           Ruler(1).ScaleLeft = ArtWork.ScaleLeft
           Ruler(1).ScaleWidth = ArtWork.ScaleWidth
     Ruler(0).Refresh
     UpdateObjects
End Sub
```

ArtWork.ScaleLeft=ArtWork.ScaleLeft+ZLeft：加上或减去 ZLeft 值，以重新确定 Artwork 图片框的内部比例。在用放大(Zoom In)或缩小(Zoom Out)功能缩放了全部控制对象后，检查以下事件是否发生，以及运行是否正确：

- 所有节点和图柄应该正确对齐，否则检查一下 Zoom 及 ZoomNode 的值。
- 当用滚动条移动桌面时，标尺线是否与移动的鼠标指针对齐？拖曳鼠标指针时，标尺标线是否对齐？如果不是这样，修改每一个 Scroll_Change 事件(清单 4.15)中的 BSM 和 SSM 值。图 5.6 显示了正确及不正确的对齐方式。

NewSW=ArtWork.ScaleWidth：存储新比例宽度和新比例高度，以便在下一个代码行中计算 PerSW 和 PerSH 的值。

PerSW=NewSW/OldSW：PerSH=NewSH/OldSH：如果在缩放之前，桌面被水平地或垂直地滚动了，PerSW(比例宽度百分比)和 PerSH(比例高度百分比)的值就被用来重新计算滚动条的当前位置。这两个值还被传至每个标尺。如果没有使用滚动条，PerSW 和 PerSH 就等于 1。

BSM=BSM * PerSW：SSM=SSM * PerSH：BSM(base Scroll movement)和 SSM(side scroll movement)的值在缩放后被用来计算每个标尺的 ScaleLeft 和 ScaleTop 属性。

Ruler(0).ScaleTop=ArtWork.ScaleTop：将每个标尺的比例重设为与 Artwork 的新比例相匹配。

UpdateObjects：按新比例重画桌面上的所有对象。

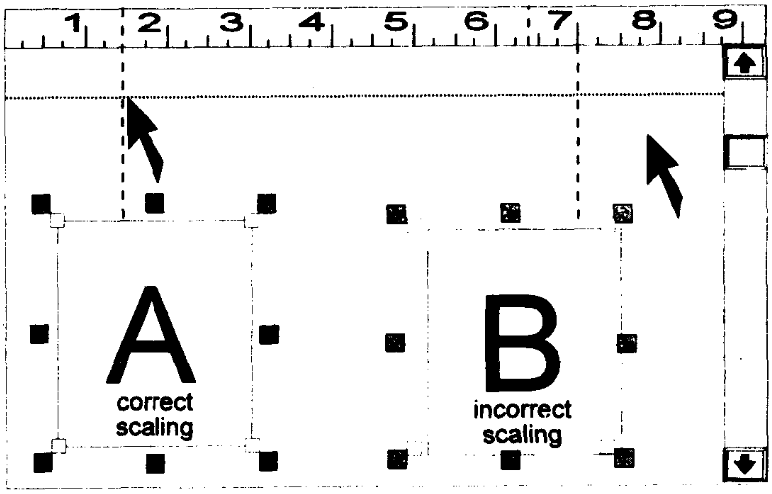


图 5.6 正确和不正确的比例缩放

5.19 本章小结

Visual Basic 语言可以将你的程序提高到新的图形化高度。通过将鼠标事件和 Line，Circle 图形语句一起使用，Artisan 程序模拟了一个程度有限的绘图包的基本部分。该 Artisan 程序并不试图成为一个功能齐全的图形包。它的主要意图是要展示图形程序的基本原理，以便能够在程序中加入一些简单的鼠标绘图及图形重新定位的技术。

还应清楚，前面章节中所用的全部过程都可用作建立用户友好程序界面的可行方法。

下面只是一些你可能想增加功能的例子：

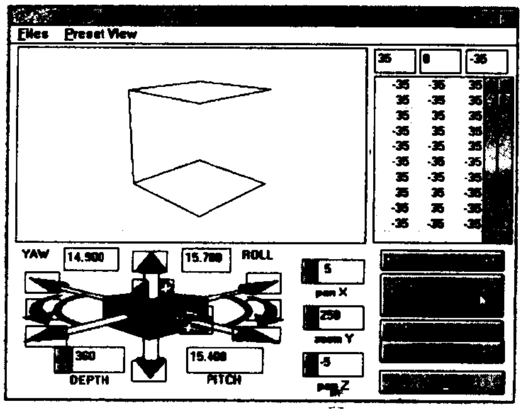
- 使用户能够用鼠标来调整直线或直方图的能力。
- 使用户能够直接在窗体或控制对象上输入文字的能力。
- 手工排列、重叠或移动其他控制对象或位图图象的能力。

应该指出的是 Visual Basic 语言的 DOS 版本有 Draw 关键字，该关键字用来连接直线对象。DOS 中的 Draw 关键字被 Visual Basic Line 语句的 Step 关键字取代，但去掉了 Angle 选项，该选项使得对象的旋转和倾斜十分容易。

Artisan 程序不能将直线或曲线对象连接到一起，并不是因为 Visual Basic 的 Line 或 Curve 语句没有连接端点的能力。Line 和 Circle 语句有 Step，Start，End 等关键字可处理它们的形状。仅用 Visual Basic 语言的图形语句绘制相连接对象的问题是不能用颜色来填充该形状。Visual Basic 语言从它的 Windows 产品中删去了 DOS Basic 语言的 Paint 语句。在 DOS Basic 语言中，Paint 语句可用来填充某个形状，例如三角形。正是在这种地方，Windows 的 API 图形库接管了原来 Basic 的功能，并提供了比任何 Basic 程序都强得多的能力。在以后的章节中，将给你介绍一些这样的 API 函数，你就可以绕过第 15 章中的 ArtAPI 程序，自己重建 Artisan 程序。还将给出另外一些建立 Bézier 曲线、连接复杂对象以及处理文字的例子。

第 6 章　Animator 程序

Animator（动画生成器）程序（见图 6.1）是一个具有框架绘制以及计算机辅助设计（CAD）功能的简单应用程序。它用于绘制出由线段连接成的简单物体。这个动画生成器（Animator program）因为其绘图能力有限，仅能处理由单一线段组成的物体框架。然而如果对绘制基本线段的过程进行修改，就能够达到更好的效果。



```
35    0    -35
-35  -35   35
 35  -35   35
 35   35   35
-35   35   35
-35  -35   35
-35  -35  -35
 35  -35  -35
 35   35  -35
-35   35  -35
-35  -35  -35
```

YAW 14.900
ROLL 15.700
DEPTH 360
PITCH 15.400
pan X 5
zoom Y 250
pan Z -5

图 6.1　Animator 程序图示

每一条线段（Line segment）的起点和终点都位于一个三维的视区（view port）中，这一视区是通过一个图片框控制对象——PictureCell 实现的，在这个图片框控制对象中使用的坐标系是二维（2-D）的，原点位于正中央。从图片框控制对象的中心（0,0）点向右移，是 X 的正方向；向左移，则是 X 的负方向。从中心向上移，是 Y 的负方向；向下移，则得到 Y 的正方向。PictureCell 的坐标示于图 6.2。

在 3-D 系统中，增加了新的一维——Z 轴。Z 轴替代了标准的 Y 轴，即垂直方向（从顶部到底部）。Y 轴则被用来表示深度，当沿 Y 轴负向增加时，相当于图象向着你运动过来；当沿 Y 轴正向增加时，则相当于图象远离你而去。图 6.3 显示了在 3-D 系统中，X，Y，Z 三轴的方向。

为了更好的理解这一新的三维坐标系，可以设想在屏幕的正中心绘制着一个人的头部，X，Y，Z 轴的原点位于头部的中央，这样，头顶就位于 Z 的负轴上，而下额则位于 Z 轴的正向；鼻子在 Y 的负轴上，后脑部位于 Y 轴正向；右脸位于 X 轴的正向而左脸位于 X 轴的负向。这样配制的坐标系是需要花点时间来掌握的。当熟悉这个坐标系以后，你就能够很轻易地绘制各种物体。

你可以打开位于目录/06ANIMAT 下的数据文件 DATABANK.DAT 来试着使用一下动画生成器。DATABANK.DAT 是一个随机存取的数据文件，其中存放着每组的 X，Y，Z 三轴的坐标值，DATABANK.DAT 中存有下列 ASCII 值：

```
-35 -35 35 ^ 35 -35 35 ^ 35 35 35 ^ -35 35 35 ^ -35 -35 35
```

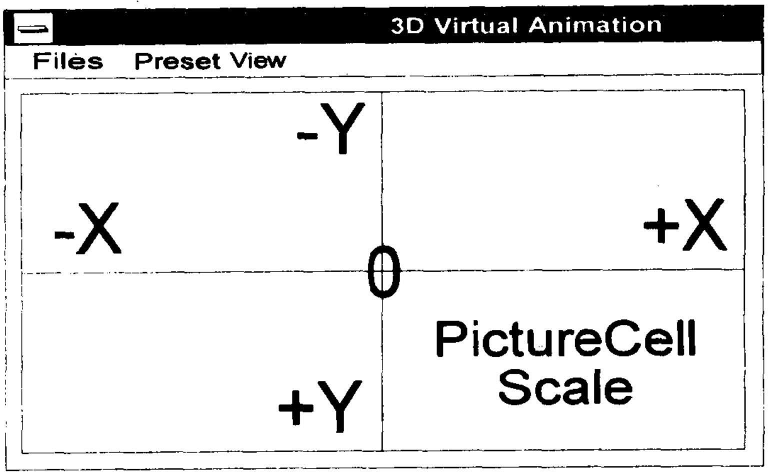

3D Virtual Animation
Files
Preset View
-Y
-X
+X
0
PictureCell
Scale
+Y

图 6.2 2-D 系统。动画生成器中使用 Y 轴表示深度，用 Z 轴替代标准的 Y 轴，X，Y，Z 三维坐标的正负方向是可变的，这样一个三维坐标系（Y 的负向指向右手方）非常适合于在较低速计算机中进行图形算法的运算

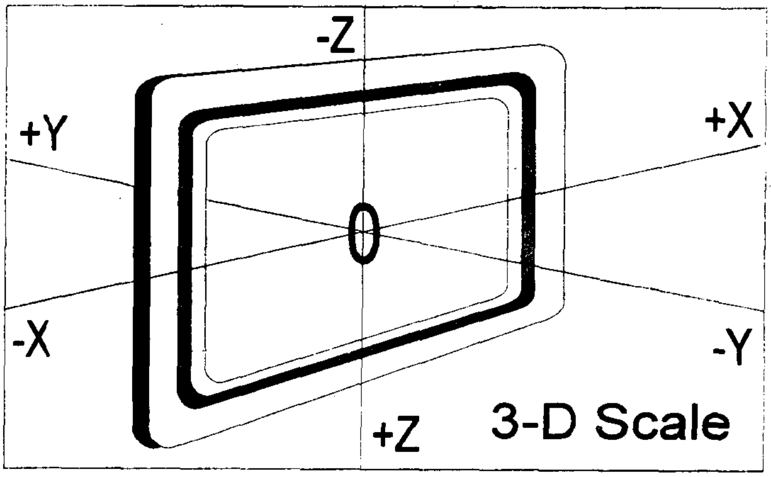

-Z
+Y
+X
0
-X
-Y
+Z
3-D Scale

图 6.3 3-D 坐标系

```
-35 -35 -35 ^ 35 -35 -35 ^ 35 35 -35 ^ -35 35 -35 ^ -35 -35 -35
```

符号 ^ 相当于空格来分开每一组不同的 X，Y，Z 值。

通过 File 菜单中的 Open 选项，打开一个数据文件以后，你可以有三种方式使物体运动起来。第一种方式是按下 Animate 按钮，物体就会按照设定的值连续的横旋（YAW）、侧旋（ROLL）或前旋（PITCH）。第二种方式是通过箭头来控制，在你选中的运动方向箭头上按下鼠标，即可使物体按照这一方向运动。第三种方式是利用滚动条来移动物体。

横旋值改变物体的水平方向，相当于绕 Z 轴旋转，图 6.4 给出了横旋的示意。

侧旋值使物体进行顺时针方向或逆时针方向的运动，图 6.5 给出了侧旋的示意。

前旋值改变了物体正面的位置，相当于绕 X 轴旋转，图 6.6 给出了前旋的示意。

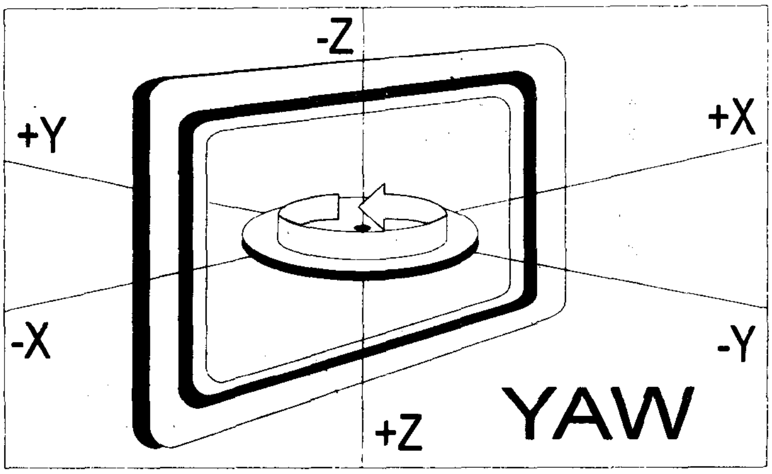


图 6.4　横旋示意图

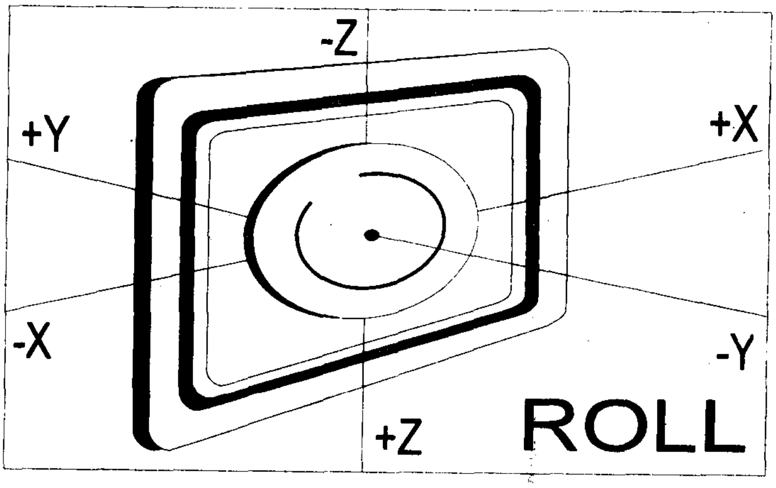


图 6.5　侧旋示意图

X 值反映了物体的宽度，Y 值反映了物体的深度，Z 值反映了物体的高度。

所有物体的旋转均为圆运动，并通过弧度来进行计算，一周为 360 度，等于 0 弧度或 6.28319 弧度，更详细的内容请见《Visual Basic 参考指南》中的 Circle 语句。

表 6.1 给出了动画生成器的窗体及控制对象的属性。

表 6.1　动画生成器窗体及控制对象的属性

Object: Form
Object Name: Viewer
Caption =3D Virtual Animation　　Height =6165
Left =75　　Top =1080
Width =8010

Object: CommandButton
Object Name: ViewButton

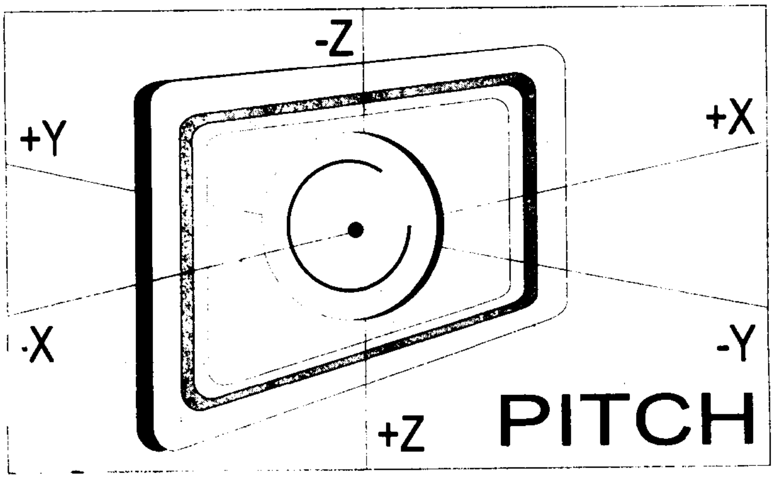


图 6.6 前旋示意图

Caption ="View Grid1"
Height =315
Left =5820
Top =3180
Width =1935

Object: Text Box
Object Name: XYZinput 3 indexed controls

Height =375
Index =0 to 3
Left =above GRID1
Top =above GRID1
Width =645

Object: Grid
Object Name: Grid1

Cols =3
FixedCols =0
FixedRows =0
Height =2595
HighLight =0' False
Left =5700
ROWS =20
ScrollBars =2' Vertical
Top =480
Width =2175

Object: Picture Box
Object Name: Bit_map

AutoSize =-1 'True
BorderStyle =0 'None
Height =2250
Left =60
Picture =Pointer.BMP
Top =3120
Width =4455

Object: CommandDialog
Object Name: CMDialog1

Left =90
Top =1665

Object: Text Box
Object Name: ValueBox 4 indexed controls

Height =405
Index =0 to 3
Left =within Bit_map
Top =within Bit_map
Width =840

Object: VscrollBar
Object Name: VScroll

Height =420
Index =3
Left =675 within Bit_map
Max =5000

SmallChange =5
Value =360
Top =1575 within Bit_map
Width =255

Object: Label
Object Name: Arrows 12 indexed controls

BackStyle =0′ Transparent
Height =225
Left =within Bit_map
Width =540
BorderStyle =1′ Fixed Single
Index =11
Top =within Bit_map

Object: Label
Object Name: Labels

Alignment =2′ Center
Height =225
Left =765
Width =960
Caption ="DEPTH"
Index =3
Top =1995

Object: Label
Object Name: Labels

Alignment =2 ′Center
Height =225
Left =2745
Width =1200
Caption ="PITCH"
Index =2
Top =1980

Object: Label
Object Name: Labels

Alignment =2 ′Center
Height =315
Left =45
Width =675
Caption ="YAW"
Index =1
Top =45

Object: Label
Object Name: Labels

Alignment =2 ′Center
Height =315
Left =3510
Width =675
Caption ="ROLL"
Index =0
Top =45

Object: CommandButton
Object Name: SingleCell

Caption ="Single Cell"
Left =5820
Width =1950
Height =360
Top =4560

Object: VScrollBar
Object Name: VScroll

Height =375
Left =4620
Min =−100
Top =4770
Index =2
Max =100
SmallChange =5
Width =255

Object: VScrollBar
Object Name: VScroll

Height =375
Left =4620
Min =100
Top =4050
Width =255
Index =1
Max =1000
SmallChange =20
Value =250

Object: Text Box

Object Name：XYZ

```
Alignment    =1 'Right Justify     Height        =375
Index        =1                    Left          =4845
Top          =4770                 Width         =735
```

Object：Text Box
Object Name：XYZ

```
Alignment    =1 'Right Justify     Height        =375
Index        =1                    Left          =4845
Top          =4050                 Width         =735
```

Object：Text Box
Object Name：XYZ

```
Alignment    =1 'Right Justify     Height        =375
Index        =1                    Left          =4845
Top          =3330                 Width         =735
```

Object：CommandButton
Object Name：ExitBtn

```
Caption      ="Exit"               Height        =360
Left         =5820                 Top           =5040
Width        =1965
```

Object：VScrollBar
Object Name：VScroll

```
Height       =375                  Index         =0
Left         =4590                 Max           =100
Min          =-100                 SmallChange   =5
Top          =3330                 Width         =255
```

Object：CommandButton
Object Name：AnimateBtn

```
Caption      ="Animate"            Height        =675
Left         =5820                 Top           =3540
Width        =1935
```

Object：CommandButton
Object Name：Pause

```
Caption      ="Pause"              Height        =315
Left         =5820                 Top           =4260
Width        =1935
```

Object：Picture Box
Object Name：PictureCell

```
Height       =3045                 Left          =120
ScaleMode    =3 'Pixel             Top           =45
Width        =5475
```

Object：Label
Object Name：Labels

```
Alignment    =2 'Center            Caption       ="pan Z"
Height       =255                  Index         =6
Left         =4620                 Top           =5175
Width        =990
```

Object：Label
Object Name：Labels

```
Alignment    =2 'Center            Caption       ="zoom Y"
Height       =315                  Index         =5
Left         =4620                 Top           =4455
```

```
     Width          =990
Object: Label
Object Name: Labels
     Alignment      =2 'Center                  Caption        ="pan X"
     Height         =315                        Index          =4
     Left           =4575                       Top            =3735
     Width          =990
Object: Menu
Object Name: MENU_F
     Caption        ="&Files"
MENU_Open
     Caption        ="&New"                     Enabled        =0 'False
     Index          =0                          Shortcut       =^ N
MENU_Open
     Caption        ="&Open"                    Index          =1
     Shortcut       =^ O
MENU_Open
     Caption        ="Save& As.."               Enabled        =0 'False
     Index          =2                          Shortcut       =^ A
MENU_Open
     Caption        ="—"                        Index          =3
MENU_Open
     Caption        ="E&xit"                    Index          =4
     Shortcut       =^ X
MENU_P
     Caption        ="&Preset View"
MENU_degrees
     Caption        ="&0 degrees"               Index          =0
MENU_degrees
     Caption        ="&45 degrees"              Index          =1
MENU_degrees
     Caption        ="&Bird's Eye"              Index          =2
```

清单 6.1 给出了动画生成器中使用的变量名。

清单 6.1　程序中使用的变量名

```
Declarations
Type AddrRecType
    XField As String * 4
    YField As String * 4
    ZField As String * 4
End Type

Global FileName As String
Global TempFile As String
Global AddrRec As AddrRecType ' An address record.
Global PINPOINTS As Integer ' Current number of records.
Global Continue As Integer
Global ButtonDown,Counter As Integer
Global NextRow As Integer
Global Untitled As String
```

ANIMATE#.BAS：全局变量，定义于 ANIMATE#.BAS 中。

AddrRec：存放着 X，Y，Z 坐标值，这些数值反映屏幕上点的位置并可达四位数字的精度。

在 ANIMATE#.BAS 文件中，为 AddrRec 设定了下面的数据结构类型(Type)：

```
Type AddrRec
    XField As String * 4
    YField As String * 4
    ZField As String * 4
End Type
```

PINPOINTS：一个随机文件中，X，Y，Z 值的起始点。

程序从数据文件中读取坐标值并通过坐标控制进行安排使你可以看到每一条线段。

Continue：标志位。用于表示动画暂停或中断后继续运动。

ButtonDown，Counter：标志位。用于表示用户输入了 Scroll 值。

NextRow：在坐标控制对象中用于表示下一行的位置。

6.1 启动时的默认值

清单 6.2 给出了启动时的默认值。

清单 6.2 启动时默认值的设定

```
Sub Form_Load ()
TempFile = "C:\Datatest.DAT"
Viewer.Caption = "3D-Viewer: " & Filename
    XYZ(0).Text = "0"
    XYZ(1).Text = "250"
    XYZ(2).Text = "0"
        ValueBox(0).Text = "05.70"
        ValueBox(1).Text = "06.30"
        ValueBox(2).Text = "05.90"
        ValueBox(3).Text = "360"
End Sub
```

TempFile：不被使用。表示一个存放数据的临时文件。

Viewer.Caption：窗体的标题栏及 3D Viewer 的临时文件名。

XYZ：启动时物体最初的摆放坐标值。

ValueBox：启动时物体最初的角度方位值。

清单 6.3 给出了使窗体位于屏幕中央的程序。

清单 6.3　使窗体位于屏幕中央

```
Sub Form_Resize ()
If WindowState = 0 Then
    Viewer. Left = (Screen. Width - Viewer. Width) \ 2
    Viewer. Top = (Screen. Height - Viewer. Height) \ 2
End If
End Sub
```

清单 6.4 给出了在退出 Animator 时如何关闭数据文件的程序。

清单 6.4　退出 Animator 及关闭数据文件

```
Sub Form_Unload (Cancel As Integer)
    Close #1
End Sub
```

图 6.7 给出了动画生成器各控制对象的名称。

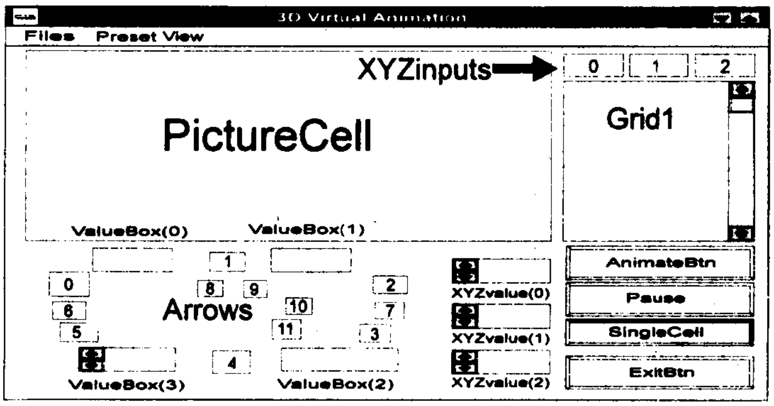

3D Virtual Animation
Files
Preset View
XYZinputs
0
1
2
PictureCell
Grid1
ValueBox(0)
ValueBox(1)
1
0
8
9
2
6
Arrows
10
7
5
11
3
4
ValueBox(3)
ValueBox(2)
XYZvalue(0)
XYZvalue(1)
XYZvalue(2)
AnimateBtn
Pause
SingleCell
ExitBtn

图 6.7　Animator 的控制对象名称

6.2　设置 3-D 动画图片框的比例

图片框控制 PictureCell 的坐标比例是可自定的。所以可以设定其原点(0,0)位于屏幕的正中心。

清单 6.5 给出了在视窗(viewport)内画一条十字交叉线的程序清单。

清单 6.5　画一条十字交叉线的程序清单

```
Sub PictureCell_Paint ()
PictureCell. Cls
X1 = PictureCell. ScaleWidth \ 2
Y1 = PictureCell. ScaleHeight \ 2
X2 = PictureCell. ScaleWidth \ 2
Y2 = PictureCell. ScaleHeight \ 2
    PictureCell. Scale (-X1, -Y2)-(X2, Y2)
```

```
X1 = PictureCell. CurrentX
Y1 = PictureCell. CurrentY
X2 = PictureCell. Width
Y2 = PictureCell. Height
PictureCell. DrawStyle = 2
    PictureCell. Line (X1, 0)-(X2, 0), QBColor(4)
    PictureCell. Line (0, Y1)-(0, Y2), QBColor(4)
SingleCell_Click
End Sub
```

X1=PictureCell ScaleWidth/2：得到图片框控制对象的正中央点，X1，X2 和 Y1，Y2 值是相同的，程序中的写法是为了清晰易看。

PictureCell. Scale(－X1，－Y1)－(X2，Y2)：定义图片框的比例以使(0，0)点位于中心。

PictureCell. Line(X1，0)－(X2，0)，QBColor(4)：在图片框中画交叉的红色线。

图 6. 8 显示了动画生成器的窗体。

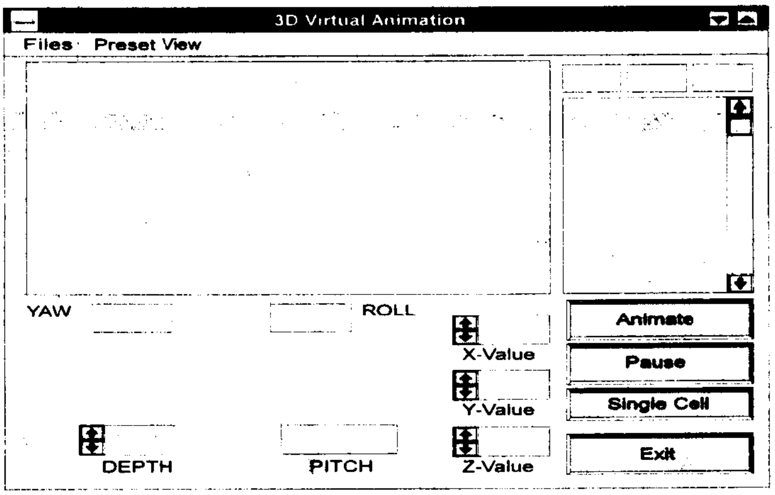

3D Virtual Animation
Files
Preset View
YAW
ROLL
X-Value
Y-Value
Z-Value
DEPTH
PITCH
Animate
Pause
Single Cell
Exit

图 6. 8 动画生成器的窗体

6. 3 开始 3-D 动画进程

在你创建或取出一个图形文件以后，可以通过按下 Animate 按钮开始一个动画过程，为了显示的需要，预制了 Movement＝0. 05 弧度作为各方向旋转的变化弧度，通过一个 while/wend 循环使物体不停地运动下去，直到你按下 Pause 按钮或 Single Cell 按钮，程序见清单 6. 6。

清单 6. 6 开始一个连续动画过程

```
Sub AnimateBtn_Click ()
If PINPOINTS = 0 Then Exit Sub
Grid1. Visible = False
Continue = True
Movement = . 05
While Continue
```

```
        Yaw = Val(ValueBox(0).Text) + Movement
        Roll = Val(ValueBox(1).Text) + Movement
        Pitch = Val(ValueBox(2).Text) + Movement
            ValueBox(0).Text = Format $ (Yaw, "00.00")
            ValueBox(1).Text = Format $ (Roll, "00.00")
            ValueBox(2).Text = Format $ (Pitch, "00.00")
        ANIMATION
        t = DoEvents()
    Wend
    End Sub
```

If PINPOINTS：当没有取到 X，Y 或 Z 的坐标值时，退出此子程序。

Continue：当 Pause 或 Single Cell 按钮以及 VScroll 控制对象被按下时，动画将停止，此处设置 Continue=TRUE，表示重新启动动画过程。

Movement=0.05：设定了横旋(YAW)；侧旋及前旋的多旋转值，当增大 Movement 的值时，物体的运动会加快，当减小 Movement 的值时，物体会运动得很慢。

注意： 假如你忘记在 YAW. Roll 或 Pitch 变量中加入"+Movement"的话，Movement 将失去作用。如程序中所示：Yaw=Val(ValueBox(0). Text)+Movement。

While/Continue：除非 Continue=False，否则继续这个循环。

YAW：横旋 ROLL：侧旋 PITCH：前旋

ValueBox(Index). Text：所有的 YAW，ROLL，PITCH 值的运算都通过这个文本框控制对象来实现。

ANIMATION：调用 ANMATION 子程序(见清单 6.7)。

t=DoEvents()：DoEvents()提供了在循环执行过程中，对键盘或鼠标输入的检查。如果在一个 while/wend 循环中不加入这一句程序，程序将死循环。

6.4 3-D 动画的计算

Animation 子程序取得 X，Y，Z 的坐标值并通过 Visual Basic 的 Line 语句画出物体的框架，每一条线段都相互连接，以组成物体的框架。所以若要绘出分离的不同物体，就需要编制新的程序。一种方法是把分离的物体用线段连接起来，在绘制这些连接线段时，绘图模式设置为 11(不可见)或将其绘图样式设置为 5(不可见)。当转向绘制数据文件中的下一个对象时，重新将绘图模式或绘图式样改为正常值。

所有的数据文件都是随机读取文件，你可以使用 write# 或 print# 语句对数据进行写入与输出。数据都以简明易见的 ASCII 字符存放于文件中，在各组 X，Y，Z 坐标值之间，可以加入逗号、空格或其他可以作为标志的字符。

清单 6.7 绘制 3-D 图形的计算程序

```
Sub ANIMATION ()
PictureCell.DrawStyle = 0
PictureCell.Cls
LastRecord = PINPOINTS:r = 0
For i = 1 To LastRecord
```

```
    Yaw = Val(ValueBox(0).Text) '5.7
    Roll = Val(ValueBox(1).Text) '6.3
    Pitch = Val(ValueBox(2).Text) '5.9
    DEPTH = Val(ValueBox(3).Text) '360
SinYaw = Sin(Yaw): CosYaw = Cos(Yaw)
SinRoll = Sin(Roll): CosRoll = Cos(Roll)
SinPitch = Sin(Pitch): CosPitch = Cos(Pitch)
Grid1.Col = 0: Grid1.Row = r: X = Val(Grid1.Text)
Grid1.Col = 1: Grid1.Row = r: Y = Val(Grid1.Text)
Grid1.Col = 2: Grid1.Row = r: Z = Val(Grid1.Text)
    YawX = (CosYaw * X) - (SinYaw * Y)
    YawY = (SinYaw * X) + (CosYaw * Y)
    RollX = (CosRoll * YawX) + (SinRoll * Z)
    RollZ = (CosRoll * Z) - (SinRoll * YawX)
    PitchY = (CosPitch * YawY) - (SinPitch * RollZ)
    PitchZ = (SinPitch * YawY) + (CosPitch * RollZ)
PanX = Val(XYZ(0).Text)
PanY = Val(XYZ(1).Text)
Zoom = Val(XYZ(2).Text)
    Xaxis = RollX + PanX
    Yaxis = PitchY + PanY
    Zaxis = PitchZ + Zoom
sx = DEPTH * Xaxis \ Yaxis
sy = DEPTH * Zaxis \ Yaxis
If r = 0 Then PictureCell.PSet (sx, sy)
        PictureCell.Line -(sx, sy), QBColor(4)
r = r + 1
Next i
End Sub
```

PictureCell.DrawStyle:默认的画笔模式——SolidPen。

LastRecord=PINPOINTS:r=0:PINPOINTS 变量表示数据文件中(X,Y,Z)坐标的总数。当这些 X,Y,Z 坐标值绘制在桌面上时,它们表示的是 3-D 空间中线段的起点和终点的位置。r 变量为网格控制对象(Grid Control)提供正确的数据——从第 0 行开始。

For i=1 To LastRecord:从 1 开始到最后一个数据均进行循环体内的处理。

YAW:横旋　ROLL:侧旋　PITCH:前旋

Depth:视界的深度感

接下来的程序计算出 YAW,ROLL 及 PITCH 的 Sin,Cos 值,通过弧度进行运算。360 度(一圈)用弧度表示是 0 弧度或 6.28319 弧度,关于弧度的知识请参见《Visual Basic 程序员指南》一书。

SinYaw:通过 sin 函数求出 YAW 的正弦值,用弧度表示。

CosYaw:通过 cos 函数求出 YAW 的余弦值,用弧度表示。

SinRoll: Roll 的正弦值,用弧度表示。

CosRoll: Roll 的余弦值,用弧度表示。

SinPitch: Pitch 的正弦值,用弧度表示。

CosPitch: Pitch 的余弦值,用弧度表示。

Grid1.Col=0:Grid1.Row=r:X=Val(Grid1.Text):(X,Y,Z)坐标值是通过每个 Grid

控制对象单元得到的。每一组(X,Y,Z)代表 3-D 空间中线段的起点或终点。

注意： X 轴代表屏幕从左至右的方向。Y 轴代表深度(屏幕上从前至后的方向),Z 轴代表从上到下的方向,详见图 6.2。

YawX=(cosYaw * X)-(sinYaw * Y):这六条语句通过三角函数的运算,得到在 3-D 空间内正确的坐标值。

PanX=Val(XYZ(0).Text):物体沿 X 轴、Y 轴或 Z 轴移动的值。

PanX:物体左右平移值。

ZoomY:物体前、后移动值。

PanZ:物体上、下移动值。

Xaxis=RollX+PanX:使物体沿 X 轴移动。

Yaxis=PitchY+ZoomY:使物体沿 Y 轴移动。

Zaxis=PitchZ+ZoomZ:使物体沿 Z 轴移动。

sx,sy:得到 Visual Basic 中 Line 函数需要的坐标值。

If r=0 Then:当 r=0 时,你可以设定最初的线段起点。

PictureCell.Line:绘制物体的每一条线段都实时地画出,并依赖于所用计算机的速度,屏幕的闪动是因为前一幅图形未被存储。

6.5 使用箭头来移动物体

当鼠标的形状为箭头的标题控制对象时,物体就会按照箭头所示的方向移动。运动的速度由变量 Increments 和 Decrements 控制,当增加 Increments 的值时,物体的运动速度会加快。

注意： Increment 值增加,物体的运动会变快。如果 Increment 值过大或计算机速度太慢,以及当屏幕的刷新速度较慢,物体的运动会有闪烁跳动感。反之物体的运动会变慢。Decrement 变量取值大小与物体运动速度之间的关系与变量 Increment 刚好相反。

清单 6.8 当箭头被单击时物体的运动

```
Sub Arrows_MouseDown (Index%, Button%, Shift%, X!, Y!)
Grid1.Visible = False
ButtonDown = True
Continue = True:Increments = -10:Decrements = 10
Select Case Index
    Case 0 'Y positive
        ZoomOut = Val(XYZ(1).Text)
        While DoEvents() And ButtonDown
        XYZ(1).Text = Str$(ZoomOut): ZoomOut = ZoomOut + Decrements
        ANIMATION
    Wend
    Case 1 'Z negitive
        PanUp = Val(XYZ(2).Text)
        While DoEvents() And ButtonDown
        XYZ(2).Text = Str$(PanUp): PanUp = PanUp + Increments
        ANIMATION
```

```
Wend
Case 2 'X positive
    PanRight = Val(XYZ(0).Text)
    While DoEvents() And ButtonDown
    XYZ(0).Text = Str $ (PanRight): PanRight = PanRight + Decrements
    ANIMATION
Wend
Case 3 'Y positive
    ZoomIn = Val(XYZ(1).Text)
    While DoEvents() And ButtonDown
    XYZ(1).Text = Str $ (ZoomIn): ZoomIn = ZoomIn + Increments
    ANIMATION
Wend
Case 4 'Z positive
    PanDown = Val(XYZ(2).Text)
    While DoEvents() And ButtonDown
    XYZ(2).Text = Str $ (PanDown): PanDown = PanDown + Decrements
    ANIMATION
Wend
Case 5 'X negative
    PanLeft = Val(XYZ(0).Text)
    While DoEvents() And ButtonDown
    XYZ(0).Text = Str $ (PanLeft): PanLeft = PanLeft + Increments
    ANIMATION
Wend
Case 6 'YAW left
    Yaw = Val(ValueBox(0).Text)
    While DoEvents() And ButtonDown
    ValueBox(0).Text = Format $ (Yaw, "00.000"): Yaw = Yaw + 0.1
    ANIMATION
Wend
Case 7 'YAW right
    Yaw = Val(ValueBox(0).Text)
    While DoEvents() And ButtonDown
    ValueBox(0).Text = Format $ (Yaw, "00.000"): Yaw = Yaw - 0.1
    ANIMATION
Wend
Case 8 'Roll negative
    Roll = Val(ValueBox(1).Text)
    While DoEvents() And ButtonDown
    ValueBox(1).Text = Format $ (Roll, "00.000"): Roll = Roll + 0.1
    ANIMATION
Wend
Case 9 'Roll positive
    Roll = Val(ValueBox(1).Text)
    While DoEvents() And ButtonDown
    ValueBox(1).Text = Format $ (Roll, "00.000"): Roll = Roll - 0.1
    ANIMATION
Wend
Case 10 'PITCH negative
    Pitch = Val(ValueBox(2).Text)
    While DoEvents() And ButtonDown
    ValueBox(2).Text = Format $ (Pitch, "00.000"): Pitch = Pitch + 0.1
    ANIMATION
```

```
        Wend
        Case 11 'PITCH positive
            Pitch = Val(ValueBox(2).Text)
            While DoEvents() And ButtonDown
            ValueBox(2).Text = Format $ (Pitch, "00.000"): Pitch = Pitch - 0.1
            ANIMATION
        Wend
    End Select
    End Sub
```

Increment 和 Decrement 这两个变量的取值使得物体在子轴方向上下平移，在 X 轴方向左右平移，在 Y 轴方向变大或缩小。

Increments：本数值被加于 XYZ TextBox 以实现沿 X，Y 或 Z 轴移动 10 个单位。

Decrements：本数值被减于 XYZ TextBox 以实现沿 X，Y 或 Z 轴移动-10 个单位。

下面的数值 0.1 被用来表示物体 Yaw，Roll 和 Pitch 值的增减：

YAW=YAW+0.1：YAW 值被加 0.1，使物体绕中心轴 Z 轴旋转+0.1 弧度。

YAW=YAW−0.1：YAW 值被减 0.1，使物体绕中心轴 Y 轴旋转−0.1 弧度。

ROLL=ROLL+0.1：ROLL 值被加 0.1，使物体绕中心轴 Y 轴旋转+0.1 弧度。

ROLL=ROLL−0.1：ROLL 值被减 0.1，使物体绕中心轴 Y 轴旋转−0.1 弧度。

PITCH=PITCH+0.1：PITCH 值被加 0.1，使物体绕中心轴 X 轴旋转−0.1 弧度。

PITCH=PITCH−0.1：PITCH 值被减 0.1，使物体绕中心轴 X 轴旋转+0.1 弧度。

图 6.9 和图 6.10 给出了表示位图中各箭头的代码值及坐标方向，图 6.11 给出了 X，Y，Z 坐标点的摆放方式。

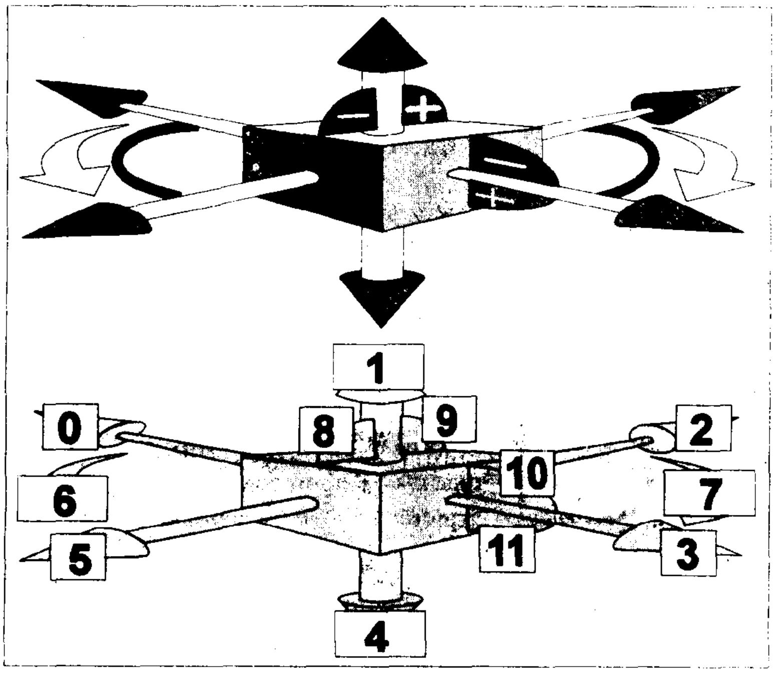


图 6.9　箭头的位图及标题控制对象的代码值

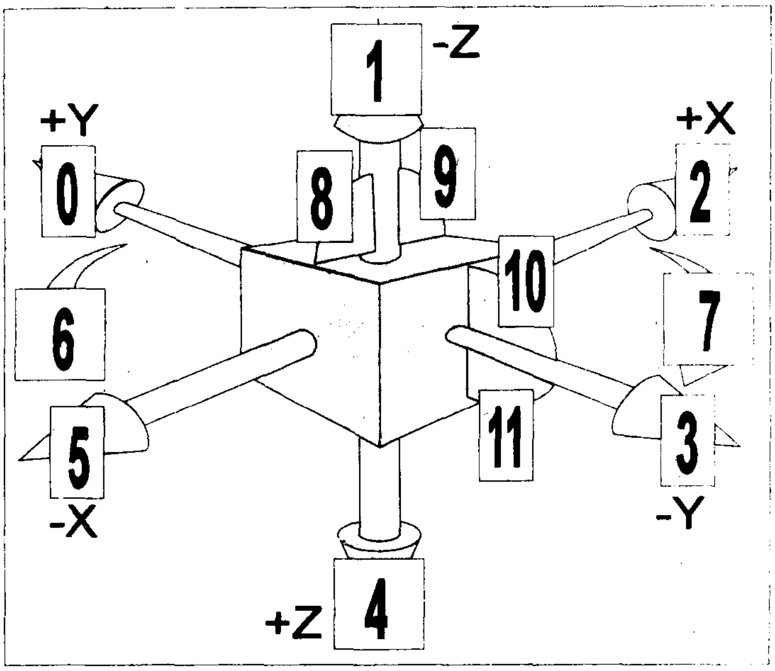

-Z
+Y
+X
-X
-Y
+Z
0
1
2
3
4
5
6
7
8
9
10
11

图 6.10 箭头的位图及坐标系

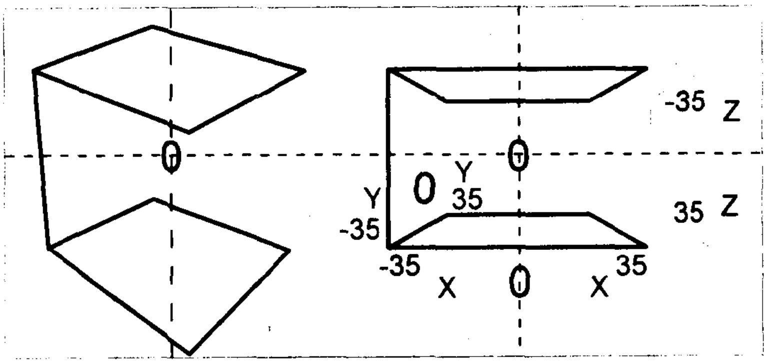

0
0
-35 Z
35 Z
Y
-35
Y
35
-35
X
0
X
35

图 6.11 X,Y,Z 值在坐标系中的位置

6.6 动画的开关控制

如清单 6.9 所示，当某个箭头上发生了 MouseDown 事件之后，在一个 Case 循环中，物体会一直运动下去，直到 ButtonDown 的值变为 False。在每个 Case 语句中，DoEvents 都被用来检测鼠标器事件，当鼠标键松开时，ButtonDown 会被置为 False 并使循环终止。

清单 6.9 动画的开关控制

```
Sub Arrows_MouseUp (Index%, Button%, Shift%, X!, Y!)
    ButtonDown = False
End Sub
```

ButtonDown：在 Arrows_MouseDown 子程序中，用作是否退出 while/wend 循环的标

志。

6.7 动画程序中的网格控制对象

网格控制对象在每一行中显示 X,Y 和 Z 的数值。把 X 的坐标值输入 Grid1 控制对象中时，只须选中 Grid1 控制对象的第一个文本框，并键入坐标值即可；按一下空格键，光标即跳到右侧的下一个文本框内。此时即可输入 Y 值，同样也可输入 Z 值。当再按空格时，将进入下一行，可以输入下一组的 X,Y,Z 坐标值了。也可以用鼠标右键直接选中某一个文本框，此时该文本框内的值被清除，你可以输入新的值。

清单 6.10 从网格控制对象中清除数据

```
Sub ClearGrid ()
    Grid1.SelStartCol = 0
    Grid1.SelStartRow = 0
        Grid1.SelEndCol = Grid1.Cols - 1
        Grid1.SelEndRow = Grid1.Rows - 1
Grid1.FillStyle = 1
Grid1.Text = ""
    Grid1.FillStyle = 0
    Grid1.HighLight = False
    NextRow = 0
End Sub
```

ClearGrid 子程序从网格控制对象中清除 X,Y,Z 的数值，变量 NextRow 重新设置 0，故 ShowData(见清单 6.18)子程序可以在最开始的行列输入新的数据，当打开一个新的数据文件时，Grid1 控制对象被清除。

Grid1-MouseDown 事件可以改变 Grid1 控制对象中单个文本框的值(见清单 6.11)，选中 Grid1 单元，使该单元对应的文本框为空即可删去 X,Y,Z 的值，或者先由键盘输入新值到相应的文本框，然后选择该列的任何一个单元。新输入的数据会被立即存入文件中，所以用户对这一点应加以注意。当然，我们也可以用 Drag 和 Drop 事件完成以上操作。

清单 6.11 改变网格单元中的数据

```
Sub Grid1_MouseDown (Button%, Shift%, X!, Y!)
If Button = 2 Then 'Right button only
    If PINPOINTS = 0 Then Exit Sub
    Continue = False: N = Grid1.Col
    If XYZInput(N).Text = "" Then XYZInput(N).Text = "0"
    If Grid1.Col = 0 Then Grid1.Text = XYZInput(0).Text:
        AddrRec.XField = XYZInput(0).Text
    If Grid1.Col = 1 Then Grid1.Text = XYZInput(1).Text:
        AddrRec.YField = XYZInput(1).Text
    If Grid1.Col = 2 Then Grid1.Text = XYZInput(2).Text:
        AddrRec.ZField = XYZInput(2).Text
End If
End Sub
```

N＝Grid1. col：N 等于你选中的 cell 的列数（0，1 或 2）。

If XYZInput(N). Text＝" "Then：假如选中的文本框为空，则置 X，Y，Z 为 0。

If Grid 1. col＝0 Then：把新的数据写入文件中。

清单 6.12 给出了怎样在一个文本框控制对象中选中所有文字的程序。

清单 6.12　以文字编辑器中所用的方式高亮显示选定的文字

```
Sub HighlightText (TextBox As Control)
    TextBox.SelStart = 0
    TextBox.SelLength = Len(TextBox.Text)
End Sub
```

TextBox. SelStart＝0：在文字的最开始插入光标。

TextBox. SelLength＝Len(TextBox. Text)：选中文字框中所有的文字。

6.8 从不同角度看一个 3-D 物体

选中 Preset View 菜单后，可以从中选择三种不同角度看一个三维物体。程序如清单 6.13 所示。

清单 6.13　Menu_degrees_Click 子程序

```
Sub MENU_degrees_Click (Index As Integer)
XYZ(0).Text = "0"
XYZ(1).Text = "250"
XYZ(2).Text = "0"
Select Case Index
    Case 0 ' 0 degrees
        ValueBox(0).Text = "00.00"
        ValueBox(1).Text = "00.00"
        ValueBox(2).Text = "00.00"
        ValueBox(3).Text = "360"
    Case 1 ' 45    degrees
        ValueBox(0).Text = "05.70"
        ValueBox(1).Text = "06.30"
        ValueBox(2).Text = "05.90"
        ValueBox(3).Text = "360"
    Case 2 ' Bird eye view
        If PINPOINTS = 0 Then Exit Sub
        BirdsEye.Show 1: Exit Sub
End Select
        SingleCell_Click
        XYZInput(0).SetFocus
End Sub
```

case 0 ' 0 degrees：置 YAW，ROLL 及 PITCH 为 0，相当于物体的正视图（从前面观察）。

case 1 ' 45 degrees：置 YAW，ROLL 及 PITCH 为启动时的缺省设置，相当于将物体的正视图偏转 45°角观察。

.case 2 'Bird eye view：打开另一个窗体，在屏幕上显示物体的正视图、侧视图和俯视图。用于显示这些视图的代码与打开数据文件时所用的代码类似。在显示物体的正视图、侧视图和俯视图时，要分别略去物体的 X，Y 或 Z 坐标值。

下面的程序给出一个物体的侧视图：

```
For i=1 To PINPOINTS
    Grid1.Col=1:Grid1.Row=r:X1=Val(Grid1.Text)
    Grid1.Col=2:Grid1.Row=r:Y1=Val(Grid1.Text)
    If r=0 Then Picture1(1).pSet(X1,Y1)
    Picture1(1).Line-(X1,Y1),QBColor(4)
    r=r+1
Next i
```

本段程序源于 Animate2 窗体中的 Picture1-Paint 子程序。如何从一个数据文件显示出图形，请参见清单 6.18。

6.9　打开或存储一个数据文件

下面的程序打开一个随机文件，并将其中存储的每组 X，Y，Z 值作为独立的数据字段，因为文件是随机存取的，故删除的部分可以作为空格存在。有关如何删除空白字段以及如何利用 Binary 访问方式或 Write# 语句改写 SaveData 过程（清单 6.15）的内容，请参阅 Visual Basic 参考手册中的随机文件部分。

最后强调一下，你应该经常使用 File 菜单下的 Save As 选项对数据文件进行备份。这个程序使用一种快速操作方式——kill 函数来删除旧的文件信息。

清单 6.14　打开和存储数据文件

```
Sub MENU_Open_Click (Index As Integer)
CMDialog1.Filter = "All Data Files (*.dat)|*.dat"
CMDialog1.Flags = &H400& Or &H1000&
CMDialog1.CancelError = True
On Error GoTo ErrorHandler
Select Case Index
    Case 0 'New
        Continue = False
        Close #1: ClearGrid
        On Error Resume Next
        Kill "C:\Datatest.dat"
        PINPOINTS = 0: PictureCell.Cls
        Viewer.Caption = "3D-Viewer: " & TempFile
    Case 1 'Open
        CMDialog1.DialogTitle = "Open"
        CMDialog1.Flags = &H1000& Or &H400& Or &H800&
        CMDialog1.DefaultExt = "dat"
        CMDialog1.Action = 1
        Close #1: ClearGrid
        Filename = CMDialog1.Filename
        Viewer.Caption = "3D-Viewer: " & Filename
        ShowData
    Case 2 'Save As...
```

```
        CMDialog1.DialogTitle - "SAVE AS..."
        CMDialog1.Flags = &H800& Or &H400& Or &H2&
        CMDialog1.DefaultExt = "dat"
        CMDialog1.Action = 2
        Close #1: TempFile = CMDialog1.Filename
        Viewer.Caption = "3D-Viewer: " & TempFile
        SaveFILE
    Case 4 'Exit
    Close #1
        End
End Select
ErrorHandler:
SingleCell_Click
Exit Sub
End Sub
```

CMDialog. Filter=:详见《Visual Basic 语言参考手册》中关于 CMDialog 控制对象中的文件标志、过滤器及错误处理等内容。

Select Case Index:选中菜单中的索引号对应的项。

Case 0:选中 New File 选项。

Continue=False:终止屏幕上进行的动画过程。

Close #1:ClearGrid:假如一个数据文件现在是打开的,则关闭之,并调用 ClearGrid 从 Grid1 控制对象中清除 X,Y,Z 的值。

On Error Resume Next:如果打开或删除文件失败,继续执行下一条语句。

Kill "C:\Datatest. dat":假如你正使用临时文件存储数据,删除之。

PINPOINTS=0:PictureCell. Cls:重置 X,Y,Z 为 0 并清除屏幕。

Case 1:选中了 Open 选项,首先执行与 Case 0 中类似的程序进行一些初始设置,然后调用 ShowData 过程(清单 6.17)打开文件。

Case 2:选中了 Save As 选项,首先执行与 Case 0 中类似的语句重设程序,然后调用 Save File 过程(清单 6.16)保存打开的文件。

Case 4:选中了 Exit 选项。

ErrorHandler:假如一个文件不能被存储或打开,退出本进程。

SingleCell_Click: SingleCell 按钮被单击,这样,新数据文件代表的 3-D 图形才能被绘出。

清单 6.15 给出怎样暂停一个动画进程。

清单 6.15 暂停动画进程

```
Sub Pause_Click ()
    Continue = False
End Sub
```

6.10 存储数据文件

清单 6.16 给出怎样将 Grid 控制对象中的数据存入文件中。

清单 6.16 将 Grid 控制对象中的数据存入文件中

```
Sub SaveDATA ()
  PINPOINTS = PINPOINTS + 1

    AddrRec.XField = XYZInput(0).Text
    AddrRec.YField = XYZInput(1).Text
    AddrRec.ZField = XYZInput(2).Text
If PINPOINTS = 1 Then Open TempFile For Random As #1 Len = Len(AddrRec)
Put #1, PINPOINTS, AddrRec
    Viewer.Caption = "3D-Viewer: " & TempFile
    XYZInput(0).SetFocus
End Sub
```

PINPOINTS：这个变量用于指示写入随机(Random)文件的位置，使用 Visual Basic 的 Put 语句在该位置存入(X,Y,Z)值。

AddrRec. XField，AddrRec. YField，AddrRec. ZField：将被存入数据文件中的字符串，它们被定义为 Type AddrRec 结构(见 ANIMATOR3.BAS 文件)，最多由 4 个字符组成，因为我们利用它们来存储屏幕上点的坐标，所以最长即为四位数。

Open：打开一个数据文件，详情请参考 **Visual Basic** 的 **Open** 语句。

Put #1：把数据写入文件，因为同一时间只能打开一个文件，所以打开的文件被称为 file #1、PINPOINTS 变量指出存入(X,Y,Z)的位置。

Viewer. Caption：窗体的标题——3-D Viewer 数据文件名。

XYZInput(0). SetFocus：将文本光标置于 Grid1 的第一个文本框上。

6.11 打开并显示数据

ShowData 过程打开数据文件并将每组 X,Y,Z 值插入 Grid1 控制对象中(见清单 6.17)。

清单 6.17 把数据从文件中读入 Grid 控制对象中

```
Sub ShowData ()
PINPOINTS = 0
If Filename <> "" Then
MENU_Open(0).Enabled = True: MENU_Open(2).Enabled = True
    Open Filename For Random As #1 Len = Len(AddrRec)
    PINPOINTS = LOF(1) / Len(AddrRec)
Else
    Exit Sub
End If
For indexpos% = 1 To PINPOINTS
    If indexpos% <> 0 Then
        Get #1, indexpos%, AddrRec
        DataXYZ$ = RTrim(AddrRec.XField) & Chr(9) & [IC:CCC]
            RTrim(AddrRec.YField) & Chr(9) & RTrim(AddrRec.ZField)
        Grid1.AddItem DataXYZ$, NewColumn
        NewColumn = NewColumn + 1
        Grid1.ColAlignment(0) = 1: Grid1.ColAlignment(1) = 1
        Grid1.ColAlignment(2) = 1
    End If
Next indexpos%
End Sub
```

PINPOINTS=0：将数据文件的字段位置设为 **0**。

If Filename：如果正在加载一个数据文件，则继续运行下面的程序。

MENU_Open：使 File 菜单的 New，Save As... 选项变灰，以便在加载一个文件或在 Grid1 中装入数据后即可使用。

Open：打开一个数据文件，详情请参考 Visual Basic 的 Open 语句。

PINPOINTS：用数据文件的总长 LOF(1)除每个记录的长度(Len(AddrRec)，得到记录的个数。

For indexpos%=1 To PINPOINTS：从第一个记录到最后一个记录，均进行循环体内的操作。

Get #1：从数据文件中得到一组(X，Y，Z)记录。

DataXYZ $：假如 X，Y 或 Z 的值不足四位，则用空格补齐，使得 X 的值显示于网格的第一列，Y 值显示于第二列，Z 值显示于第三列。

Grid1.AddItem：将(X，Y，Z)插入 Grid1 中并显示。

NewRow：增加一条记录，以便加入新的(X，Y，Z)值。

Grid1.ColAlignment：使数字靠右侧显示，以便于阅读。

Next indexpos%：重复上述过程直至所有数据均显示于 Grid1 控制对象中。

6.12 文件中数据的更新

SingleCell 按钮用于将新的 X，Y，Z 值置入 Grid1 控制对象中，程序见清单 6.18。

清单 6.18 单幅图形显示

```
Sub SingleCell_Click ()
If PINPOINTS = 0 Then Exit Sub
    Continue = False
    PictureCell.Cls
    ANIMATION
    Grid1.Visible = True
End Sub
```

IF PINPOINTS=0 Then：无数据文件调入，则退出进程。

Continue：停止正在运行的动画进程。

PictureCell.Cls：清屏。

ANIMATION：调用 ANIMATION 进程(清单 6.7)绘制单幅图形。

6.13 通过滚动条移动 3-D 物体

物体的 X，Y，Z 三轴坐标显示于 XYZ 文本框内。按动文本框旁边的滚动条移动物体，可以改变文本框中的坐标值。单击箭头时，随着物体沿箭头方向运动，文本框内显示的 X，Y，Z

值也会产生相应的变化（清单 6.19）。

清单 6.19 改变图形的轴

```
Sub VScroll_Change (Index As Integer)
Select Case Index
    Case 0
        XYZ(0).Text = Format $ (VScroll(0).Value)
    Case 1
        XYZ(1).Text = Format $ (VScroll(1).Value)
    Case 2
        XYZ(2).Text = Format $ (VScroll(2).Value)
    Case 3
        ValueBox(3).Text = Format $ (VScroll(3).Value)
End Select
End Sub
```

case 0：3-D 物体左右移动。

case 1：3-D 物体前后移动。

case 2：3-D 物体上下移动。

XYZ(i).Text：通过滚动条控制对象，改变物体的 X，Y，Z 轴值。

清单 6.20 给出使文本高亮显示的程序。

清单 6.20 使文本高亮显示

```
Sub XYZinput_GotFocus (Index As Integer)
    i = Index
    HighlightText XYZInput(i)
End Sub
```

Highlight Text XYZInput(i)：将位于 Grid1 上方三个文本框中显示的 X，Y，Z 坐标值高亮显示，以便对这些输入值进行清除。

6.14 输入新的数据

XYZInput 文本框被用来输入新的数据，在 Grid1 控制对象中，有很多文本框需键入 X，Y 或 Z 值，你可以通过按 Tab 键由一个文本框跳入另一个文本框（清单 6.21）。

注意：在最后一个文本框中键入空格，会使全部数据存入文件中。

清单 6.21 通过按空格键存入数据

```
Sub XYZinput_KeyPress (Index As Integer, KeyAscii As Integer)
If KeyAscii >= Asc("0") And KeyAscii <= Asc("9") Or KeyAscii = &H20 Or [IC;CCC]
    KeyAscii = &H8 Or KeyAscii = Asc("-45") Then   Else KeyAscii = 0: Beep
If KeyAscii = &H20 Then 'spacebar
    Index = Index + 1
        If Index > 2 Then
```

```
                Index = 0
                DataXYZ $  = XYZInput(0).Text & Chr(9) & XYZInput(1).Text & [IC:CCC]
                    Chr(9) & XYZInput(2).Text
                Grid1.AddItem DataXYZ $ , NextRow
                NextRow = NextRow + 1
                SaveDATA
    XYZInput(0).Text = "": XYZInput(1).Text = "": XYZInput(2).Text = ""
            End If
    XYZInput(Index).SetFocus
    KeyAscii = 0: MENU_Open(0).Enabled = True: MENU_Open(2).Enabled = True
    End If
    End Sub
```

If KeyAscii>=Asc("0")：在文本框内输入数字。

If KeyAscii=&H20：如果键入的是空格，则跳入 Grid1 控制对象中的下一文本框内。

If Index>2 Then：当 Index>2 时，数据就会被加入 Grid1 控制对象中。

DataXYZ $：保存 X，Y 和 Z 的输入值，并在每个值后加一个制表符 Chr(9)。

Grid1. Add Item：将每个 X，Y 和 Z 值插入 Grid1 控制对象中以便观察。

NewRow：继续操作 Grid1 控制对象的下一行。

SaveData：调用此例程来把新的 X，Y 和 Z 值保存到数据文件中。如果不希望将数据写入文件则可删去此行。

XYZinput：清除所有输入 X，Y 和 Z 值的文本框。

XYZinput(i). SetFocus：将文本光标设到第一个输入框，以便继续添加数值(如果希望继续的话)。

KeyAscii：置 KeyAscii 为 0，清除缓冲区，以便重新键入新值。

MENU_Open. Enabled：使 File 菜单中的 New，Save As... 等选项可被使用。

6.15 本章小结

Animator 程序很容易设计，并使你能够绘制出很生动的三维模型。

你应该设法将第 2 章到第 5 章介绍的绘图工具引入 Animator，这样就可容易地用鼠标器来绘制 3-D 模型，而不需费力地输入坐标。

你可以试着改变一下在 Animator 过程中使用的 sin 和 cos 值，你就会得到按椭圆轨迹运动的动画；改变、转换或插入 Line 函数的不同偏移量会使 3-D 模型的不同线段相互交叉，产生特别的效果；也可以试着改变线段的颜色，从 QBColor(4)改到 QBColor(Rnd * 14)，这样可以观察每条线段的运动。

把第 1 章中使用的 TestMove 程序中的图形插入，即使 Line 函数由以下程序代替：

```
If r=0 Then PictureCell.PSet(sx,sy)
PictureCell.Line-(sx,sy),QBColor(4)
PictureCell.Line-(sx,sy)-Step(100,0+N)
PictureCell.Line-Step(0,100)
PictureCell.Line-Step(-100,0-N)
PictureCell.Line-Step(30,50)
PictureCell.Line-Step(-30,-150)
PictureCell.Circle(X1+50,Y1+50),25
```

改变 AnimateBtn_Click 事件中的 Movement 变量的值，试验一下该值的变化对动画效果的影响。

最后，你可能会发现动画效果有一点闪烁而且屏幕更新的速度比较慢，对这一问题还没有什么快速便捷的解决办法。因为基于 Visual Basic 的绘图速度大体只能如此，如果采用 API 函数的话可能会更慢，在实时状态下，DOS 的 Basic 语言程序只能达到如此的动画程度。近期 Windows 提出了新的办法。微软(Microsoft)提供了两个新的 API 库——Win32G 和 WinG32.DLL。适用于动画及游戏设计，可以参见随书所配磁盘 16 MISC 目录下 WinG API 函数的例子 WinGame 程序，为了使用新的 WinG 函数，你必须装载 WinG SDK Kit。WinG SDK Kit 可从 Microsoft WinMM 库中免费获得，其中包括所有需要的 .DLL 文件。

第 7 章 多重查找和替换

本章涉及到文本串的处理部分。文本串可以作为诸如文字说明以及桌面排版等图形应用程序的一个组成部分。本书主要讲述 Visual Basic 的图形功能(如直线、曲线等),但本书还是将多重查找和替换程序包括进来并作为单独一章,为的是可以让读者能够洞悉 Visual Basic 的串处理方法和函数。在这个程序中调整文本位置的例子显示了如何使用 Visual Basic 的 TextHeight 和 TextWidth 函数来精确对齐处在图片框控制对象中的一行文本。在第 12 章中,将介绍更多的使用 Windows API 文本函数的文本属性。

当一个按钮被按下的时候,多重查找和替换程序可以处理任何 ASCII 文本文件并可替换多个词。程序完成这些工作只采用了一个简单的循环,就是读取查找列表框控制对象的每一个条目,然后使用 Visual Basic 的选择文本(Sel)语句将其高亮显示并使用保存着替换列表的相邻列表框的头一个条目来改变文本。

在 ReplaceCom 过程(见清单 7.17)末尾的 Do/Loop 语句导致了多重查找和替换的不同。当起始位置(startpos)变量等于 0 时,光标位置就已到达了文件的末端。然后本过程选取查找列表框中的下一个条目,继续循环直到所有条目都依照 CodeLst2(替换)列表框被替换掉。

为 Visual Basic 3.0 版或更高版本设计的多重查找和替换程序使用了 CMDialog.VBX 来保存或打开文本文件。使用多重查找和替换程序时,操作步骤如下:

1. 在 Search For 文本框中键入一个单词。
2. 按一下位于窗体左侧列表框下方的 Add 命令按钮,或按一下回车键。
3. 对 Replace With 文本框重复前面步骤,这时应使用窗体右侧的 Add 命令按钮。你可以用左右箭头键来为每一个文本框设置焦点。

注意: 如果你经常使用相同的查找、替换列表,可以使用列表框的 ADDITEM 方法把每一个列表条目加进 Form_Load 事件中。例如:

```
CodeLst1.AddItem "Company".
    CodeLst2.AddItem "Co.".
```

使用 Kill One Listing 按钮可以删除列表框中你高亮显示的一个查找/替换项。

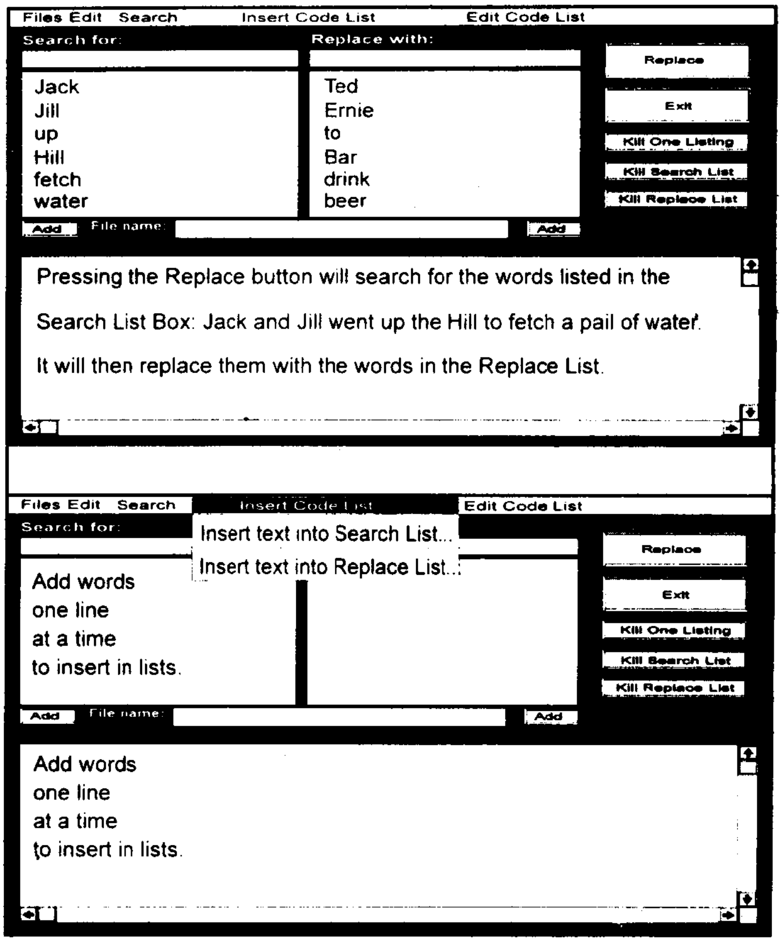
使用 Kill Search List 和 Kill Replace List 按钮可以从相应列表框中删除整个列表。

使用 Replace 按钮来启动对一个文本文件的查找和替换操作。查找替换结束后,使用 File 菜单项中 Save As 项来保存文件。

查找和替换程序的窗体和控制对象属性列于表 7.1 中。

表 7.1 查找和替换程序窗体和控制对象的属性

Object: Form
Object Name: replace


Files Edit Search
Insert Code List
Edit Code List
Search for:
Replace with:
Replace
Exit
Kill One Listing
Kill Search List
Kill Replace List
Add
File name:
Add
Insert text into Search List...
Insert text into Replace List...

Jack
Jill
up
Hill
fetch
water

Ted
Ernie
to
Bar
drink
beer

Pressing the Replace button will search for the words listed in the Search List Box: Jack and Jill went up the Hill to fetch a pail of water.

It will then replace them with the words in the Replace List.

Add words
one line
at a time
to insert in lists.

Add words
one line
at a time
to insert in lists.

图 7.1　窗体及其控制对象名

Caption = "MULTI-SEARCH&REPLACE"　Top = 1080
Height = 7275　Width = 9690
Left = 105

Object: Frame
Object Name: Frame1

BackColor = &H00FF0000&　Top = 90
Height = 3480　Visible = 0 'False
Left = 90　Width = 9045

Object: CommandDialog
Object Name: CMDialog1

Left = 8370
Top = 2925

Object: CommandButton
Object Name: AddBtn

Caption = "Add"　Top = 2970
Height = 285　Width = 735
Index = 1
Left = 6435

Object: CommandButton
Object Name: AddBtn

Caption =″Add″
Height =285
Index =0
Left =90
Top =2970
Width =735

Object：CommandButton
Object Name：replacebtn
Caption =″Replace″
Height =645
Left =7470
Top =135
Width =1815

Object：CommandButton
Object Name：Exitbtn
Caption =″Exit″
Height =645
Left =7470
Top =855
Width =1815

Object：CommandButton
Object Name：Deletebtn
Caption =″Kill One Listing″
Height =375
Index =0
Left =7470
Top =1575
Width =1800

Object：CommandButton
Object Name：Deletebtn
Caption =″Kill Search List″
Height =375
Index =1
Left =7470
Top =2025
Width =1800

Object：CommandButton
Object Name：Deletebtn
Caption =″Kill Replace List″
Height =375
Index =2
Left =7470
Top =2475
Width =1800

Object：Text Box
Object Name：SearchBox
Height =315
Left =90
Top =270
Width =3480

Object：Text Box
Object Name：ReplaceBox
Height =315
Left =3690
Top =270
Width =3480

Object：List Box
Object Name：codelst1
Height =2370
Left =90
Top =585
Width =3480

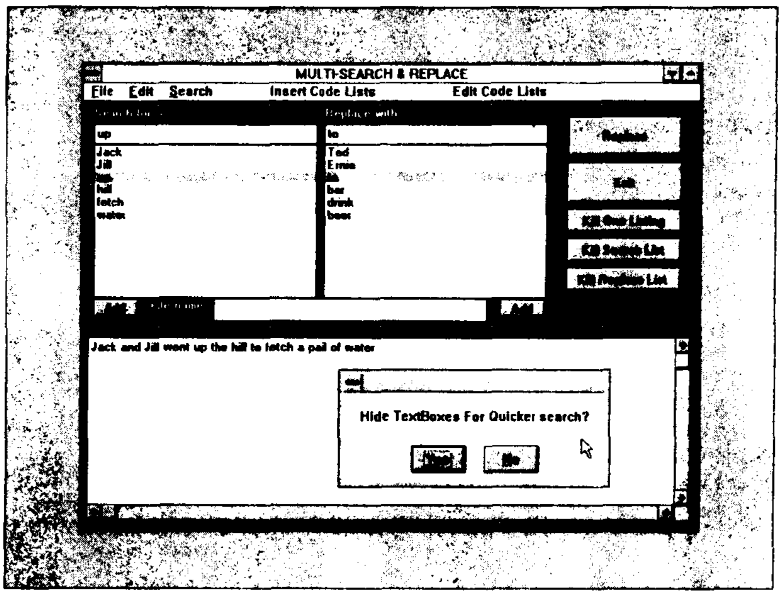
Object：List Box
Object Name：codelst2
Height =2370
Left =3690
Top =585
Width =3480

Object：Label
Object Name：Labels

```
      BackColor      =&H00FF0000&                  Index      =2
      Caption        ="File name:"                 Left       =990
      ForeColor      =&H00FFFFFF&                  Top        =2970
      Height         =255                          Width      =990

Object: Label
Object Name: Labels
      BackColor      =&H00FF0000&                  Index      =1
      Caption        ="Replace with:"              Left       =3690
      ForeColor      =&H00FFFFFF&                  Top        =0
      Height         =255                          Width      =1215

Object: Label
Object Name: Labels
      BackColor      =&H00FF0000&                  Index      =0
      Caption        ="Search for:"                Left       =90
      ForeColor      =&H00FFFFFF&                  Top        =0
      Height         =255                          Width      =1215

Object: Label
Object Name: FileNameLabel
      Height         =330                          Top        =2970
      Left           =1980                         Width      =4245

Object: Text Box
Object Name: Textbox
      Height         =2895                         ScrollBars =3 'Both
      Index          =0                            Top        =3660
      Left           =90                           Visible    =0 'False
      MultiLine      =-1 'True                     Width      =9375

Object: Menu
Object Name: Menu_File
      Caption        ="&File"

Menu_FileSub
      Caption        ="Open"
      Index          =0

Menu_FileSub
      Caption        ="Save As..."
      Index          =1
      Shortcut       ={F2}

Menu_FileSub
      Caption        ="File Manager"
      Index          =2
      Shortcut       =^F

Menu_FileSub
      Caption        ="-"
      Index          =3

Menu_FileSub
      Caption        ="Quit"
      Index          =4
      Shortcut       =^Q

Menu_Edit
      Caption        ="&Edit"
```

```
Menu_EditSub
    Caption         ="Cut"
    Index           =0
    Shortcut        =^X
Menu_EditSub
    Caption         ="Copy"
    Index           =1
    Shortcut        =^C
Menu_EditSub
    Caption         ="Paste"
    Index           =2
    Shortcut        =^V
Menu_EditSub
    Caption         ="—"
    Index           =3
Menu_EditSub
    Caption         ="Clear NotePad"
    Index           =4
Menu_Search
    Caption         ="&Search"
Menu_FindSub
    Caption         ="Find"
    Index           =0
Menu_FindSub
    Caption         ="Find Selected Text"
    Index           =1
    Shortcut        =^S
Menu_FindSub
    Caption         ="Replace"
    Index           =2
    Shortcut        =^R
Menu_Insert
    Caption         ="Insert Code Lists"
Menu_InsertSub
    Caption         ="Insert Text into Search List..."
    Index           =0
Menu_InsertSub
    Caption         ="—"
    Index           =1
Menu_InsertSub
    Caption         ="Insert Text into Replace List..."
    Index           =2
Menu_EditCodes
    Caption         ="Edit Code Lists"
Menu_EditCodeSub
    Caption         ="Edit Search List with NotePad"
    Index           =0
Menu_EditCodeSub
```

```
    Caption        ="—"
    Index          =1
Menu_EditCodeSub
    Caption        ="Edit Search List with NotePad"
    Index          =2
```


MULTI-SEARCH & REPLACE
File
Edit
Search
Insert Code Lists
Edit Code Lists
Search for
Replace with
up
Jack
Jill
hill
fetch
to
Ted
Ernie
bar
drink

Jack and Jill went up the hill to fetch a pail of water

Hide TextBoxes For Quicker search?

图 7.2　查找(Search For)和替换(Replace with)列表框

7.1　本程序中用到的变量

清单 7.1　在启动时声明的变量名

```
Declarations
Const BUFFER_SIZE = 16384
Dim Buffer As String ' file I/O buffer
Dim GlobalFilename As String
```

Const Buffer_Size: 用在 Opencom 菜单事件中，调一个文本文件到主文本框控制对象中。在 Do/While 循环中，每次读取包含 16,384 个字符的文本块直到文件结束。增大这个数值将导致缓冲区溢出，减小这个值会导致闪烁效果。Visual Basic 的文本框控制对象具有略少于 64KB 的字符存储区。

Buffer As String: 也被用在 OpenCom 事件中，用于临时存储从文本文件中取到的每一个文本块。当整个文件都在缓冲区中时，此文件内容就被传送到主文本框并显示出来。

Dim GlobalFilename As String: 保存着你打开的文本文件名。

7.2　在启动时为控制对象定位

Form_Resize 事件(列于清单 7.2 中)根据窗体的大小适当地重新定位所有控制对象。

通过使用框架控制对象进行分组，只需很少语句即可移动较多的控制对象。

清单 7.2　重新定位窗体中的控制对象

```
Sub Form_Resize ()
Pixel = Screen.TwipsPerPixelX
Pixel = Screen.TwipsPerPixelY
If WindowState = 1 Then Exit Sub
    Frame1.Top = 5
    Frame1.Left = 5
    Frame1.Width = (Replace.Width - 200) \ Pixel
TextBox(0).Top = Frame1.Height + 10
TextBox(0).Left = 5
TextBox(0).Height = (Replace.Height \ Pixel) - (Frame1.Height + 60)
TextBox(0).Width = (Replace.Width - 200) \ Pixel
Frame1.Visible = True
TextBox(0).Visible = True
End Sub
```

Pixel：将高度和宽度从屏幕单位(twip)转换成象素单位。对于大多数 VGA 型监视器来说，变量 Pixel 等于 15。

If WindowsState=1 Then Exit Sub：如果窗体处于极小化状态，则退出此过程。

7.3　给代码列表加入条目

这里有两个标题为 Add 的按钮。第一个按钮的索引号为 0，用来把键入到 Search for 文本框中的文本插入到对应的查找(Search)列表框。第二个按钮的索引号为 1，用来把键入到 Replace with 文本框中的文本插入到相应的替换(Repalce)列表框中。

清单 7.3　向列表框中插入查找或替换条目

```
Sub Addbtn_Click (Index As Integer)
  Select Case Index
    Case 0
        If SearchBox.Text = "" Then Exit Sub
        CodeLst1.AddItem SearchBox.Text
        SearchBox.Text = ""
        SearchBox.SetFocus
    Case 1
        If ReplaceBox.Text = "" Then Exit Sub
        CodeLst2.AddItem ReplaceBox.Text
        ReplaceBox.Text = ""
        ReplaceBox.SetFocus
    End Select
End Sub
```

Select Case Index：根据你所选择的按钮，执行特定索引号的 Case 块。

Case 0：查找(Search)列表框下面的 Add 按钮被选中。

If SearchBox：如果查找文本框中没有文本，则退出此事件。

CodeLst1. AddItem：把显示于 Search for 文本框中的文本加入到 CodeLst1 列表框（即查找列表框）。

SearchBox. Text=""：Search for 文本框中文本被清除。

SearchBox. SetFocus：重新把文本光标（caert）置于 Search for 文本框。

Case 1：替换列表框下面的 Add 按钮被选中。

If ReplaceBox：如果替换文本框中没有文本，则退出此事件。

CodeLst2. AddItem：把显示于 Replace with 文本框中的文本加入到 CodeLst2 列表框（即替换列表框）。

ReplaceBox. Text=""：Replace with 文本框中内容被清除。

ReplaceBox. SetFocus：重新把文本光标（caret）置于替换文本框。

Addbtn_GotFocus 事件使对应于正在编辑的列表框的 Add 按钮成为默认按钮。这样就可以更加方便地向列表框中插入条目（只要按回车键即可）。

清单 7.4　在按钮间切换焦点

```
Sub Addbtn_GotFocus (Index As Integer)
    Select Case Index
        Case 0
            addbtn(0). Default = True
        Case 1
            addbtn(1). Default = True
    End Select
End Sub
```

addbtn(0). Default=True：使当前 Add 按钮成为默认按钮。当回车键被按下时，Addbtn_Click 事件（见清单 7.3）就会被触发。

7.4　在列表框中选择条目

CodeLst1 列表框中的某个词必须与 CodeLst2 列表框中的一个词相对应。否则，查找和替换列表中的条目将不匹配，并且 CodeLst1_Click 事件（见清单 7.5）将会发出信息以通知用户改正某一列表。

清单 7.5　用一个替换词来匹配一个查找词

```
Sub codelst1_Click ()
If CodeLst1. ListCount <> CodeLst2. ListCount Then
    MsgBox "ONE OF YOUR LISTS IS LONGER THAN THE OTHER!"
    Exit Sub
End If
    SearchBox. Text = CodeLst1. Text
    ReplaceBox. Text = CodeLst2. Text
    CodeLst2. ListIndex = CodeLst1. ListIndex
    CodeLst2. SetFocus
End Sub
```

If CodeLst1. ListCount：如果查找或替换列表不等长，一个警告信息将被显示。

SearchBox. Text：将高亮条目显示于查找文本框。

ReplaceBox. Text：在替换文本框中寻找匹配条目。

Codelst2. ListIndex. SetFocus：高亮显示匹配的查找和替换条目。

Codelst2_Click 列表框事件（见清单 7.6）类似于 Codelst1_Click 事件（清单 7.5），不同的是高亮显示替换条目而非查找条目。

清单 7.6 用一个查找词来匹配一个替换词

```
Sub codelst2_Click ()
If CodeLst1. ListCount <> CodeLst2. ListCount Then
    MsgBox "ONE OF YOUR LISTS IS LONGER THAN THE OTHER!"
    Exit Sub
End If
SearchBox. Text = CodeLst1. Text
    ReplaceBox. Text = CodeLst2. Text
    CodeLst1. ListIndex = CodeLst2. ListIndex
    CodeLst1. SetFocus
End Sub
```

If Codelst1. ListCount：如果查找与替换列表不等长，则显示一条警告信息。

SearchBox. Text：将高亮条目显示于查找框中。

ReplaceBox. Text：寻找匹配索引条目并显示于替换框中。

Codelst1. ListIndex，SetFocus：高亮显示匹配的替换及查找条目。

7.5 从列表框中删去条目

Delete_Btn_Click 事件（列于清单 7.7）定义了可用来删去列表框中条目的三个按钮的实现方式。按钮 Kill One Listing 删去高亮的一对查找与替换条目。按钮 Kill Search List 删去查找列表框中所有条目。按钮 Kill Replace List 删去替换列表框中的所有条目。

清单 7.7 从列表中移去并删除一个条目

```
Sub Deletebtn_Click (Index As Integer)
Select Case Index
    Case 0
        If CodeLst1. Text = "" Or CodeLst2. Text = "" Then Exit Sub
        N = CodeLst1. ListIndex
        CodeLst1. RemoveItem (n)
        CodeLst2. RemoveItem (n)
        SearchBox. Text = ""
        ReplaceBox. Text = ""
    Case 1
        CodeLst1. Clear : SearchBox. Text = ""
    Case 2
        CodeLst2. Clear : ReplaceBox. Text = ""
End Select
End Sub
```

Select Case Index：根据用户所选择按钮的索引号，执行相应的Case块。

Case 0：选择了Kill One Listing按钮。

If CodeLst1. Text：如果当前没有条目则退出本事件。

N：表示两个列表框中选中（高亮）条目的索引号。

CodeLst1：RemoveItem：删去高亮条目。

CodeLst2：RemoveItem：删去高亮条目。

SearchBox，ReplaceBox：删去文本框中所有字符。

Case 1：选择了Kill Search List按钮。

Codelst1. Clear：删去了Search for列表框中所有条目。

Case 2：选择了Kill Replace List按钮。

Codelst2. Clear：删去了Replace with列表框中所有条目。

7.6　用于编辑文本的工具

这一节讲述的是各种可用在菜单选择项之下的编辑工具。当用户在Search For文本框中输入一些字符后，可以使用Search菜单项来从一个文本文件中寻找一个匹配的字串。可以使用Find Selected Text选项来继续一次一个地查找单词。这两个菜单选项都使用了FindCom过程（见清单7.8）来执行其功能。

清单7.8　在文本中查找一词

```
Sub FindCom ()
Static curindex As Long
Dim Find As String
    Find = SearchBox. Text
    If Find = "" Then Exit Sub ' nothing to find
    curindex = InStr(curindex + 1, TextBox(0). Text, Find)
        If curindex > 0 Then
            TextBox(0). SelStart = curindex - 1
            TextBox(0). SelLength = Len(Find)
            TextBox(0). SetFocus
        Else
            MsgBox "No More Matching Strings"
        End If
End Sub
```

Static curindex：通过使用InStr函数来保存文本中的光标位置。

Find As String：临时保存正在查找的文本。

If Find="" Then：如果Search for文本框中无文本则退出此过程。

curindex=InStr：（当前索引位置）InStr函数返回查找框内文本第一次出现的位置，然后利用这一位置继续下一次查找。

Instr：INSTR函数是如下定义的：

1. 文本中开始查找的起点位置；
2. 正在查找的文本的拷贝；

3. 正在查找的单词。

If curindex>0 Then：这表明已找到了一个匹配单词，所以要将它高亮显示。

TextBox(0). SelStart：把光标置于单词的正前方。

TextBox(0). SelLength：将单词高亮显示。

TextBox(0). SetFocus：将焦点置于文本框以便进行重新输入、删除或编辑。

Else：已经没有更多匹配字串，显示一条信息。

7.7 编辑列表框中条目

可以使用 Insert Code Listing 和 Edit Code Listing 菜单项来编辑任何查找或替换列表。也可以往主文本框中输入一个完整的查找或替换列表，然后把所有条目一次性插入某个列表框中。你必须每行输入一个条目并加上一个回车换行符(可以用按回车键来达到目的)。

清单 7.9 编辑一个查找或替换列表

```
Sub Menu_EditCodeSub_Click (Index As Integer)
Dim CL As String
Select Case Index
    Case 0 'Edit Search Code fields
        TextBox(0). Text = ""
    For i = 0 To CodeLst1. ListCount
        CL = CodeLst1. List(i)
        TextBox(0). Text = TextBox(0). Text + CL + Chr$ (13) + Chr$ (10)
    Next i
        CodeLst1. Clear
        SearchBox. Text = ""
    Case 2 'Edit Replace Code fields
        TextBox(0). Text = ""
    For i = 0 To CodeLst2. ListCount
        CL = CodeLst2. List(i)
        TextBox(0). Text = TextBox(0). Text + CL + Chr$ (13) + Chr$ (10)
    Next i
            CodeLst2. Clear
            ReplaceBox. Text = ""
End Select
    TextBox(0). SetFocus
    SendKeys "%{HOME}"
End Sub
```

Dim CL As String：保存列表框文本条目。

Select Case Index：依赖于你所希望编辑的列表框来进行后续工作。

Case 0：编辑查找列表框。

TextBox(0). Text=""：清除主文本框中任何输入的字符或打开的文件的内容。

For i=0 To CodeLst1. ListCount：对 CodeLst1 中从索引号 0 开始到最后一个条目执行后续工作。

TextBox(0). Text=TextBox(0). Text：除了每一行单独处理外，这跟打开一个文件完全一样。你也许希望使用一个缓冲器来防止文本闪烁现象。只要加入一个回车和换行符

(CR/LF)来使文本框中每一个列表条目基本等长即可。

Case 2:编辑替换列表框。

Case 2 的语句块和 Case 1 的情况基本相同,只不过它处理的是 CodeLst2 列表框,而是 CodeLst1 列表框。

7.8　怎样剪切、拷贝和粘贴文本

可以用 Edit 菜单项来对一个文本文件进行剪切、拷贝和粘贴操作。所有功能都用到了 Windows 的剪贴板(Clipboard)特性。

清单 7.10　将文本剪切、拷贝或粘贴到记事本(notepad)上

```
Sub Menu_EditSub_Click (Index As Integer)
Select Case Index
    Case 0 'Cut
        clipboard.SetText Screen.ActiveControl.SelText
        Screen.ActiveControl.SelText = ""
        SendKeys "{HOME}", 1
    Case 1 'Copy
        If Screen.ActiveControl.SelText <> "" Then
        clipboard.SetText Screen.ActiveControl.SelText, CF_TEXT
        End If
    Case 2 'Paste
        Screen.ActiveControl.SelText = clipboard.GetText()
    Case 4 'Clear TextBox
        TextBox(0).Text = ""
End Select
End Sub
```

Select Case Index:取决于用户希望执行何种编辑操作,然后执行相应 Case 块。

Case 0:将高亮文本剪切到剪贴板。

clipboard.SetText:使用 Screen.ActiveControl 来获知从查找、替换或主文本框的哪一个来剪切文本。Screen.ActiveControl 属性可以告知剪贴板哪个文本控制对象具有当前焦点,以使剪贴板可以对这个文本框控制对象进行剪切、拷贝或粘贴。

SendKeys:置文本光标到 HOME 位置。

Case 1:拷贝高亮文本到剪贴板。

Clipboard.SetText:同剪切类似(Case 1),但是通过在 SelText 后面加上 CF_TEXT 指定了是拷贝一个文本格式。

Case 2:将保存在剪贴板中的文本粘贴到一个文本框中。

Screen.ActiveControl.SelText:GetText 关键字可以恢复任何剪切或粘贴自剪贴板的文本。

7.9　打开或保存一个文本文件

清单 7.11 中的 Menu_FileSub_Click 函数通过 File(文件)菜单项来打开或保存显示于主文本框中的文本文件。Open(打开)菜单项调用 Opencom 过程(见清单 7.14),Save As...

(以……保存)项调用 SaveCom 过程(见清单 7.18)。你也可以切换到 Windows 文件管理器(File Manager)以检查文件的状态。

清单 7.11　打开或保存文件

```
Sub Menu_FileSub_Click (Index As Integer)
Select Case Index
    Case 0 'Open
        'Opencom
    Case 1 'Save As
        SaveCom
    Case 2 'File Manager
        Dummy = Shell("Winfile")
    Case 4 'Quit
        End
End Select
End Sub
```

Select Case Index:根据所选择的文件菜单项条目,执行相应的 Case 块。

Case 0:打开文件。

Opencom:调用 Opencom 例程来显示 Openfile 框。

Case 1:以新文件名保存文件。

Savecom:调用 Savecom 过程来显示一个输入框。

Case 2:转到 Windows 文件管理器。

Dummy=Shell("WinFile"):启动 Windows 文件管理器以观察文件是否已保存好。

清单 7.12 中的 Menu_FindSub_Click 事件是用来寻找 Search for 文本框中的文本或在主文本框中高亮显示的一个词。第三个选项完成对列表中所有条目的查找和替换。

清单 7.12　在文本中寻找另一段文字

```
Sub Menu_FindSub_Click (Index As Integer)
Select Case Index
    Case 0 'Find
        FindCom
    Case 1 'Find Highlighted
        If TextBox(0).SelText <> "" Then
            SearchBox.Text = TextBox(0).SelText
        End If
        FindCom
    Case 2 'Replace
        ReplaceCom
End Select
End Sub
```

Select Case Index:根据选择的菜单项,执行与菜单项索引号相对应的 Case 块。

Case 0:在主文本框中寻找与 Search for 文本框中文本相匹配的任何文本或输入信息。

FindCom:调用此过程以用来从文本中找出正在查找的单词。

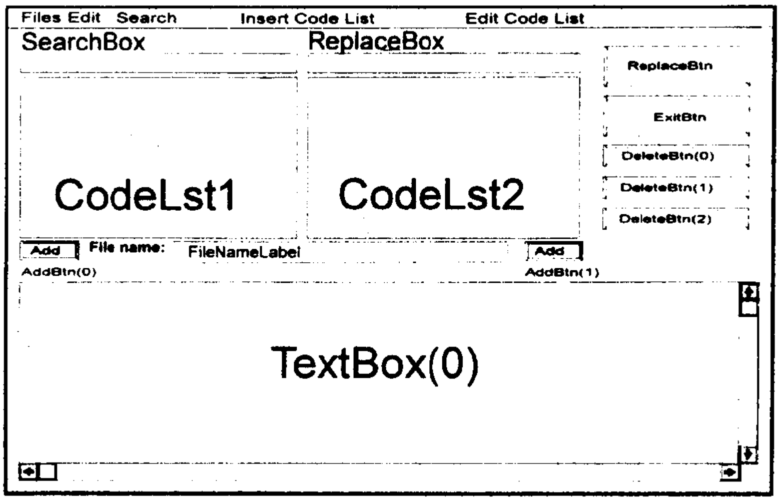

Files Edit Search
Insert Code List
Edit Code List
SearchBox
ReplaceBox
ReplaceBtn
ExitBtn
DeleteBtn(0)
DeleteBtn(1)
DeleteBtn(2)
CodeLst1
CodeLst2
Add
File name:
FileNameLabel
Add
AddBtn(0)
AddBtn(1)
TextBox(0)

图 7.3　窗体控制对象名的说明

Case 1：希望找到主文本框中高亮部分的每一次出现位置。

If TextBox(0).Text：如果主文本框中没有高亮文本，默认取为显示于 Search for 文本框中的文本。

SearchBox.Text：如果此段文本在主文本框中被高亮显示，则寻找这个字串在当前文件中的每一次出现。

FindCom：调用此过程以用来从文本中找出正在查找的单词。

Case 2：查找并替换所有查找和替换列表中的条目。这和按下窗体中较大的 Replace（替换）按钮的效果是一样的。

Menu_Insert_Click 事件使你可以获取列于记事本（Notepad）文本框中的词并将之插入到查找列表框（Code Lst1）或替换列表框（Code Lst2）之一。如果你有一个很大的查找或替换单词列表，你可以像在 Windows 记事本文件中一样编辑这些单词，然后把它们插入到合适的列表框中。

清单 7.13　把主文本框中的每一行文本插入一个列表框

```
Sub Menu_InsertSub_Click (Index As Integer)
If Not Right $ (TextBox(0).Text, 1) = Chr $ (10) Then
MsgBox "YOU MUST HAVE A SINGLE RETURN AT END OF TEXT!": Exit Sub
End If
Dim word As String
    Select Case Index
    Case 0 'Insert into Search field
    While Not TextBox(0).Text = ""
        EndofWord% = InStr(1, TextBox(0).Text, Chr $ (10))
        word = Left $ (TextBox(0).Text, EndofWord%)
        TextBox(0).SelStart = 0
        TextBox(0).SelLength = Len(word)
        codes $ = Left $ (TextBox(0).SelText, EndofWord% - 2)
        CodeLst1.AddItem codes $
        TextBox(0).SelText = ""
    Wend
        Case 2 'Insert into replace field
    While Not TextBox(0).Text = ""
        EndofWord% = InStr(1, TextBox(0).Text, Chr$(10))
        word = Left$(TextBox(0).Text, EndofWord%)
        TextBox(0).SelStart = 0
        TextBox(0).SelLength = Len(word)
        codes$ = Left$(TextBox(0).SelText, EndofWord% - 2)
        CodeLst2.AddItem codes$
        TextBox(0).SelText = ""
Wend
End Select
End Sub
```

If Not Right $ (Text. Box(0). Text,1)：本例程使用换行符(Chr $ (10))来识别主文本框中每一文本行的结尾。为使本例程正常工作，文件行后必须加入换行符。

MsgBox：通知用户通过按回车键来加入最后一个换行符(Chr $ (10))。

Word：代表要插入列表的单词或文本行。

Select Case Index：取决于你希望将主文本置入哪一个列表框。与菜单项索引号匹配的Case块将被执行。

Case 0：将主文本插入查找列表框。

While Not：循环直到Notepad文本框中所有文本均已移去并插入列表框中。

EndofWord%：表示每一文本行中换行符的位置。InStr函数可以得到整个文件、要查找的文本及要寻找字符的起始位置。

Word：代表主文本框中每一行单词或字符串。

在以下叙述中，将要高亮显示每一文本行并提取出来。

TextBox(0). SelStart：置文本光标于本行最前面。

TextBox(0). SelLength：高亮显示整行。

codes $：保存上一步选择或高亮显示的文本。

现在将去掉每行中的回车符和换行符。一种方法就是使用Visual Basic的Left $语句。Left $语句有两个参数。第一个是已选择或高亮显示的文本。第二个是本行除去回车和换行符的字符个数。

CodeList1. AddItem：将选中或高亮显示的文本行插入列表框。

TextBox(0). SelText：清除选中文本为下一行作准备。

Wend：重复循环，处理下一文本行。

Case 2：将主文本插入替换列表框。除了将每一文本行插入替换列表框外，Case 2部分与Case 0基本相同。

7.10 打开一个文本文件

为Visual Basic 3.0或更高版本设计的查找和替换程序使用了CMDialog.VBX来得到

文件名。

清单 7.14　以二进制方式打开一个文件

```
Sub Opencom ()
Dim F As String
Dim BufNum As Integer
CMDialog1.Filter = "All Files (*.*)|*.*|Text Files(*.txt)|*.txt"
CMDialog1.Flags = &H1000& Or &H800&
CMDialog1.CancelError = True
On Error GoTo BadFile
CMDialog1.FilterIndex = 2
CMDialog1.Action = 1
GlobalFilename = CMDialog1.Filename
    If GlobalFilename <> "" Then
        FileNameLabel.Caption = GlobalFilename
    Else
        Exit Sub
    End If
    F = FileNameLabel.Caption
    If F = "" Then Exit Sub
        On Error GoTo BadFile
    BufNum = FreeFile ' get next free buffer number
    Open F For Binary As #BufNum
    TextBox(0).Text = ""
    Do While Not EOF(BufNum)
        Buffer = Space$(BUFFER_SIZE)
        Get #1, , Buffer
        TextBox(0).Text = TextBox(0).Text + Buffer
    Loop
    Close #BufNum
    TextBox(0).Text = RTrim$(TextBox(0).Text) ' trim trailing spaces
' * * * * * * * * * * * Error-handler * * * * * * * * * *
BadFile: Beep
    Exit Sub
End Sub
```

Dim F As String：代表 FilenameLabel.Caption，即为文件名。

Dim BufNum As Integer：代表每一文件的唯一文件号。

CMDialog1：请参考《Visual Basic 语言指南》中的“公共对话框控制对象”和“过滤筛选器”部分，可找到 CMDialog 控制对象的方法与属性值的说明。

If GlobalFilename：当你通过 OpenFile 窗体或 CMDialog.VBX 得到文件名后，文件名及路径都保存在一个全局文件名变量中。如果没有选择文件就关闭了 OpenFile 窗体，全局文件名就不会被设置，并导致必须退出本例程。

If F="" then Exit Sub：没有文件被选择，退出例程。

On Error Goto BadFile：当错误发生时，转向标有 BadFile 的程序行。这是一个很短的错误处理程序，它将不会返回诸如“驱动器门打开”或“计算机故障”之类的信息。它所做的只是在重置 OpenBox.Caption 后让你再试一次。

Open F For Binary As #BufNum：按二进制方式打开所选择的文件，并赋给该文件一

个唯一的文件号(通常为 1)。

TextBox(0). Text="":清除主文本框中可能存在的文本。

Do While Not Eof (BufNum):执行以下例程直到到达文件尾部(EOF)。

Buffer=Space $ (BufferSize):代表一个保留的缓冲空格区,其长度为 Buffer_Size。每一块文本都在此缓冲区中处理。

Get #1,Buffer:从文件中读取一块文本。

TextBox(0). Text:将文本块置入主文本框中。

Loop:重复操作直到整个文本文件都进入主文本框。

Close #BufNum:清除文件缓冲区内存,关闭文件。

TextBox(0). Text:去掉文件尾部可能出现的任何多余空格。

Error Handler:如果无法打开文件,计算机会发出"嘟嘟"声。可添加附加的错误处理代码以满足你的特殊要求。

Beep:发出警告响声并退出此过程。

7.11 查找并替换条目

从清单 7.15 到 7.17 是有关查找和替换列表框中的所有条目的清单。

清单 7.15 激活某列表框的 Add(添加)按钮

```
Sub ReplaceBox_GotFocus ()
    addbtn(1).Default = True
End Sub
```

addbtn(1). Default:可以通过按回车键来插入一新条目。

清单 7.16 中的 ReplaceBtn_Click 事件询问是否希望隐藏记事本。

清单 7.16 选择了 Replace 按钮

```
Sub replacebtn_Click ()
If TextBox(0).Text = "" Then Exit Sub
    Msg = "Hide TextBoxes For Quicker search?"
    Response = MsgBox(Msg, 4, TextBoxes)
    If Response = 6 Then
        TextBox(0).Visible = False
    End If
ReplaceCom
    TextBox(0).Visible = True
    MsgBox "NO MORE MATCHING STRINGS", 64, "INFORMATION"
    TextBox(0).SetFocus
    SendKeys "^ {HOME}"
End Sub
```

If TextBox(0). Text:如果记事本文本框中无文本则退出本例程。

在查找和替换的时候,隐藏主文本框会使处理一长列查找和替换单词更快一些。一个可

视的文本框会为每一个正查找的词将文本光标(caret)置于词前面(SelStart),然后将此段文本高亮(SelLength)并在需要时滚屏。图形用户接口(GUI)将不得不为可视屏幕作这些工作。此外,文本框滚动条垂直和(或)水平滚动一个长的文本文件更使得操作过程非常之慢。为了克服这些缺点,可以在执行代码时隐藏主文本框。

msg:询问用户是否隐藏主文本框。

Response:显示带 Yes 和 No 按钮的消息。

If Response=1:当选择 Yes 时隐藏主文本框。

TextBox(0). Visible:隐藏主文本框。

ReplaceCom:调用本程序的核心——ReplaceCom 过程。

TextBox(0). Visible:如果文本框已被隐藏,则显示它。

当替换按钮被按下时,ReplaceCom 过程(见清单 7.17)执行实际的一个或多个词的查找和替换。

清单 7.17　查找和替换的主要代码

```
Sub ReplaceCom ()
Dim Find As String, Replace As String
If CodeLst1.ListCount <> CodeLst2.ListCount Then Exit Sub

For i = 0 To CodeLst1.ListCount - 1
Do
    Find = CodeLst1.List(i)
    SearchBox.Text = Find
    Replace = CodeLst2.List(i)
    ReplaceBox.Text = Replace
    startpos = InStr(startpos + 1, TextBox(0).Text, Find)

    If startpos = 0 Then Exit Do
    TextBox(0).SelStart = startpos - 1
    TextBox(0).SelLength = Len(Find)
    TextBox(0).SelText = Replace

    Loop While startpos > 0

Next i
End Sub
```

If CodeLst1. ListCount:如果两个列表框不等长,则退出此过程。

For i=0 To CodeLst1. ListCount-1:对查找列表框中每一条目重复下一语句块。

Do:重复以下语句块。

Find:代表 CodeLst1(查找)列表框中的一个单词或一段文本。

SearchBox. Text:在查找过程中显示每个词。

Replace:代表 CodeLst2(替换)列表框中的一个单词或一段文本。

ReplaceBox. Text:在替换过程中显示每个词。

startpos=InStr:使用 InStr 函数来保存每一个正在查找的串的起始位置。

下面是对 InStr 函数的说明:

startpos+1:InStr 函数开始每次查找的第一个字符位置。

Textbox(0).Text：被查找的文本文件。

Find：要寻找的单词。

If startpos=0 Then Exit Do：表明已查找到文件尾部，并且单词 Find 已被成功替换。这时退出 Do/Loop 循环。这里有一个技巧，就是把这个语句放在后续的 SelText 语句之前。这样就可以保证在文本不存在时，不出现错误信息。

TextBox(0).SelStart：将文本光标定位于被查找单词之前。

TextBox(0).SelLength：选择或高亮显示正在查找的单词。

TextBox(0).SelText：用替换文本替换查找文本。

Loop While startpos>0：继续寻找单词 Find 直到到达文件尾部。

Next i：对查找和替换列表框中下一对词重复操作。

7.12 保存文本文件

列表清单 7.18 的 SaveCom 过程可保存所有出现在主文本框中的文本。

清单 7.18 SaveCom 过程以二进制方式保存文件

```
Sub SaveCom ()
Dim BufNum As Integer
Dim i As Long

CMDialog1.Filter = "All Files (*.*)|*.*|Text Files(*.txt)|*.txt"
CMDialog1.Flags = &H2& Or &H800&
CMDialog1.CancelError = True
On Error GoTo BadFileName
CMDialog1.FilterIndex = 2
CMDialog1.Action = 2
GlobalFilename = CMDialog1.Filename
    If GlobalFilename <> "" Then
        FileNameLabel.Caption = GlobalFilename
    Else
        Exit Sub
    End If
    If (GlobalFilename <> "") Then
        On Error GoTo BadFileName
        BufNum = FreeFile ' get next free buffer number
        Open GlobalFilename For Binary As #BufNum

        For i = 1 To Len(TextBox(0).Text) Step BUFFER_SIZE
            Buffer = Mid$ (TextBox(0).Text, i, BUFFER_SIZE)
            Put #BufNum, , Buffer
        Next i
        Close #BufNum
    End If
Exit Sub
' ****************** Error-Handler ****************
BadFileName:
    Exit Sub
End Sub
```

Dim BufNum As Integer：代表唯一的文件名。

Dim i As Integer：表示要保存的文本块。

CMDialog1：请参考《Visual Basic 语言指南》的“公共对话框控制对象”和“过滤筛选器”等章节，可找到有关 CMDialog 控制对象的方法和属性值的说明。

GlobalFilename：代表要保存的文件名及路径。

If (GlobalFileName<> "")：如果路径及文件名存在，则可继续保存文件。

On Error Goto BadFileNum：如果在保存文件过程中发生了错误，跳转到 BadFileName 程序行（错误处理部分）。

BufNum：取得下一个可用文件号。

Open：以二进制方式打开文件，并赋以唯一文件号。文件名及路径由用户选择。

For i=1 To Len(TextBox(0).Text) Step Buffer_Size：保存文件。可以 Buffer_Size 为单位分块保存。

Buffer：代表要保存的每一块文本。Mid $ 函数包括以下内容：

(Textbox(0))：Mid $ 函数查看的文本。

(i)：每一个文本缓冲区的起始位置。

(Buffer_Size)：放入缓冲区的文本数量。

Put #BufNum,,Buffer：将每一块文本保存至文件。

Next：重复操作，保存下一块文本。

Close #BufNum：文件已保存完毕，将它关闭。

7.13　文本间隔

多重查找和替换程序的文本间隔示例涉及到 Visual Basic 的 TextWidth 和 TextHeight 属性。当选中标有“Text Spacing example”（文本间隔示例）的菜单项时，一个图形框控制对象将填满整个窗体区域。将鼠标置于图片框中任何位置都将会自动地以正确的行间隔坐标来放置文本光标。下列菜单项与控制对象是用来实现文本间隔功能的：

1. Menu_txt：标有“Text spacing example”的菜单项。
2. Picture1：一个包含着本示例所有控制对象的图片框控制对象。
3. BUIF：一个用来格式化输入文本的图片框控制对象（如正常、斜体及粗体等）。
4. Label1：一个用来模拟文本光标的标号控制对象。
5. Timer1：使文本光标闪烁的定时器控制对象。

清单 7.19 中的 Menu_txt 事件将窗体变成一个功能有限的小型文本编辑器。

清单 7.19　改变窗体的外观

```
Sub Menu_txt_Click ()
    Picture1.Cls 'erase old text images
    Picture1.Move 0, 0, ScaleWidth, ScaleHeight
    Menu_Edit.Visible = Not Menu_Edit.Visible
    Menu_EditCodeS.Visible = Not Menu_EditCodeS.Visible
    Menu_File.Visible = Not Menu_File.Visible
    Menu_Insert.Visible = Not Menu_Insert.Visible
    Menu_Search.Visible = Not Menu_Search.Visible
    Picture1.Visible = Not Picture1.Visible
    If Picture1. Visible Then
        Caption = "This is a Picture Box Control"
    Else
        Caption = "MULTI-SEARCH & REPLACE"
    End If
End Sub
```

Picture1. Cls：清除任何旧的文字图象。

Picture1. Move：使图片框控制对象和窗体一样大。

Menu_Edit. Visible：禁止所有被查找和替换示例使用的菜单条目。

清单 7.20 中的 Picture1_MouseUp 事件利用 TextHeight 和 TextWidth 方法来计算正确的文本光标位置。

清单 7.20 计算文本的行距

```
Sub Picture1_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Timer1. Enabled = True
Linecount = Y \ Picture1. TextHeight("")
    X1 = X
    Y1 = Picture1. TextHeight("") * Linecount
    Y2 = Picture1. TextHeight("")
    Label1. Move X1, Y1, 30, Y2
    Label1. Visible = True
    Picture1. CurrentX = X1: Picture1. CurrentY = Y1
End Sub
```

Timer1. Enabled=true：启动计时器以使文本光标（实际上是一标题控制对象）可以闪烁。

Linecount=Y\Picture1. TextHeight("")：计算图片框中可容纳多少行文本。

X1=X：用来把文本光标（标题控制对象）置于鼠标光标处。

Y1=Picture1. TextHeight("") * Linecount：用行的高度（包括了行距）乘以行数可以把文本光标置于正确的 Y 坐标处。

Y2=Picture1. TextHeight("")：文本的高度（包括行距）将使文本光标具有正确高度。

Label1. Move X1,Y1,30,Y2：将文本光标置于正确的文本行坐标处。

Picture1. CurrentX=X1：修改图片框的 CurrentX 与 CurrentY 以反映文本光标的位置。

清单 7.21 中的 Picture1. KeyPress 事件将显示输入的每个字符并将文本光标前移到下一个字符位置。

清单 7.21 向前移动文本光标位置

```
Sub Picture1_KeyPress (KeyAscii As Integer)
Label1. Visible = True: Timer1. Enabled = False
    If Chr(KeyAscii) = Chr(13) Then
        Y1 = Y1 + Picture1. TextHeight("")
        Picture1. CurrentY = Y1 'line feed
        Picture1.CurrentX = X1 'return
        Label1.Move X1, Y1, 30, Y2
        Exit Sub
    End If
    CursorX = Picture1.CurrentX         'save CurrentX
    Picture1.Print Chr$(KeyAscii) 'forces CurrentX to 0
    CharWidth = Picture1.TextWidth(Chr$(KeyAscii))
    Picture1.CurrentX = CursorX + CharWidth: Picture1.CurrentY = Y1
    Label1.Move Picture1.CurrentX, Y1, 30, Y2
    Timer1.Enabled = True
End Sub
```

Timer1.Enabled=False：在输入文本时应关闭定时器以使光标不再闪烁。

If Chr(KeyAscii)=Chr(13) Then：当回车键被按下时，CurrentY 进到下一行，同时 CurrentX 与第一行文本左对齐。文本光标（标题控制对象）就移到下一行的位置，同时结束本次输入键事件。

CursorX=Picture1.CurrentX：保存 CurrentX 坐标以便在后面的 Print 语句执行后可以恢复光标位置。在默认情况下 Print 语句同按下回车键一样将 CurrentX 强制设为 0。

Picture1.Print Chr$(KeyAscii)：打印字符。

CharWidth=Picture1.TextWidth(Chr$(KeyAscii)：取得刚输入的字符的宽度。

Picture1.CurrentX=CursorX+CharWidth：向前移动的距离为刚输入的字符的宽度。

Label1.Move：重新设置文本光标到新的字符位置。

Timer1.Enabled=True：打开定时器使文本光标可以闪烁。

清单 7.22 中的 BIUF_MouseDown 事件将改变当前字体属性（如正常体、斜体、黑体等等）。

清单 7.22　改变字形

```
Sub BIUF_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
    X = Fix(X)
    BIUF.Line (X, 0)-(X + 1, 1), , BF
''''''''''''''
Select Case X
    Case 0    'Bold
        Picture1.FontBold = Not Picture1.FontBold
        B_Button = Not B_Button
    Case 1    'Italic
        Picture1.FontItalic = Not Picture1.FontItalic
        B_Button = Not B_Button
    Case 2    'Underline
        Picture1.FontUnderline = Not Picture1.FontUnderline
        B_Button = Not B_Button
End Select
End Sub
```

BIUF.Line(X,0)-(X+1,1),,BF：反显位图图象以使按钮看上去是被按下的。

Select Case X：因为 BUIF 图形被规格为 4 个单位宽，所以可用 X 坐标来匹配 Case 块。

这样当选择一个按钮时，就将切换字体的黑体、斜体和下划线属性。

清单 7.23 中的 BUIF_MouseUp 事件将显示 CMDialog 控制对象，这样就可以改变字体和其他字符属性（如大小、颜色等）。

清单 7.23 显示字体对话框

```
Sub BIUF_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    X = Fix(X)
        If Fix(X) = 3 Then
        BIUF.Line (X, 0)-(X + 1, 1), , BF
        CMDialog1.Flags = &H100 Or &H1& Or &H200& Or &H40000
        CMDialog1.FontName = Picture1.FontName
        CMDialog1.FontSize = Picture1.FontSize
        CMDialog1.FontBold = Picture1.FontBold
        CMDialog1.FontItalic = Picture1.FontItalic
        CMDialog1.FontUnderline = Picture1.FontUnderline
        CMDialog1.FontStrikethru = Picture1.FontStrikethru
        CMDialog1.Color = Picture1.ForeColor
        CMDialog1.Action = 4 'font dialog box
        Picture1.FontName = CMDialog1.FontName
        Picture1.FontSize = CMDialog1.FontSize
        Picture1.FontBold = CMDialog1.FontBold
        Picture1.FontItalic = CMDialog1.FontItalic
        Picture1.FontUnderline = CMDialog1.FontUnderline
        Picture1.FontStrikethru = CMDialog1.FontStrikethru
        Picture1.ForeColor = CMDialog1.Color
DoEvents
    If Picture1.FontBold = True Then B_Button = True Else B_Button = False
    If Picture1.FontItalic = True Then I_Button = True Else I_Button = False
    If Picture1.FontUnderline = True Then U_Button = True Else U_Button = False
    If B_Button = True Then BIUF.Line (0, 0)-(1, 1), , BF
    If I_Button = True Then BIUF.Line (1, 0)-(2, 1), , BF
    If U_Button = True Then BIUF.Line (2, 0)-(3, 1), , BF
End If
        Picture1.SetFocus
        Picture1.CurrentX = Label1.Left
End Sub
```

CMDialog：请参考《Visual Basic 语言指南》中的“公共对话框控制对象”和“过滤筛选器”章节，可找到有关 CMDialog 字体控制对象的方法与属性值的说明。

If B_Button=True Then：重新设置所有反显的按钮以匹配当前字体属性。

清单 7.24 中的 BUIF_Paint 事件将绘图模式置于反显方式（6），并把图片框大小调整为 4 个单位宽。

清单 7.24 调整 BUIF 图片框

```
Sub BIUF_Paint ()
    BIUF.DrawMode = 6 'Invert
    BIUF.Scale (0, 0)-(4, 1)
```

```
End Sub
```

BIUF.DrawMode：反转图片框的绘制模式。

BIUF.Scale：将图片框的内部坐标调整为 4 个单位宽，1 个单位高。

清单 7.25 中的 Timer1_Timer 事件来回切换文本光标（标题控制对象）的可见属性。文本光标即因此而闪烁。

清单 7.25　使文本光标闪烁

```
Sub Timer1_Timer ()
    If Picture1.Visible = True Then
Label1.Visible = Not Label1.Visible
    End If
End Sub
```

Label1.Visible：切换标题控制对象的可视属性。

7.14　本章小结

多重查找和替换程序使用了 Visual Basic 的很大一部分串操作方法和函数，并且在你自己的程序中进行串操作测试时可以交叉引用这些方法和函数。在使用多重查找和替换程序时，你最有可能保存的是刚在其上执行过查找和替换的文本文件。此外，还可以将可能经常使用的查找列表或替换列表单独保存下来。如果经常使用相同的列表，可以在启动部分中加入代码以自动设置好这个列表。比如替换一些需要缩写的词如 company，limited，state（州名）和 province name（省名），可替换为 Co.，Ltd.，CA. 和 Ont.。

文本间隔示例演示了怎样在一个图片框控制中轻而易举放置文本行。你也许希望加入一些功能来模拟一个小型字处理器，这就很可能需要使用 Windows API 函数来精细调整行距、字距和版面对齐等事项。第 12 章中的三个文本对齐程序演示了 Visual Basic 可用的一些 API 文本函数。

下面的例子显示了一个可能用来自动将文本光标（标题控制对象）置于显示字符之间的方法。它需要一个被声明为 Dim tBuf(100) As String 的全局数组。

这里就是 Picture1_MouseUp 事件的一个例子：

```
    Timer1.Enabled = True
Linecount = Y \ Picture1.TextHeight("")
    X1 = X
    Y1 = Picture1.TextHeight("") * Linecount
    Y2 = Picture1.TextHeight("")
For R=0 To 99
  If Fix(X)>=tBuf(R) And Fix(X)<=tBuf(R)+(tBuf(R+1)-tBuf(R))\2 Then
    Label1.Move tBuf(R), Y1, 30, Y2
    FoundChar=True:Exit For
  ElseIf Fix(X) >= tBuf(R)+(tBuf(R+1)-tBuf(R))\2 And Fix(X)<=(tBuf(R+1))
Then
    Label1.Move tBuf(R+1), Y1, 30, Y2
    FoundChar=True:Exit For
  End If
Next R
  If FoundChar=False Then Label1.Move X1,Y1,30,Y2
    Label1.Visible = True
```

下面是 Picture1_KeyPress 事件的一个例子：

```
Static CursorX As Integer
Label1.Visible = True: Timer1.Enabled = False
    If Chr(KeyAscii) = Chr(13) Then
        Y1 = Y1 + Picture1.TextHeight("")
        Picture1.CurrentY = Y1 'line feed
        Picture1.CurrentX = X1 'return
        Label1.Move X1, Y1, 30, Y2
        Exit Sub
    End If
Static R As Integer
    CursorX = Picture1.CurrentX          'save CurrentX
    tBuf(R)=Fix(Label1,Left):R=R+1
    Picture1.Print Chr$ (KeyAscii) 'forces CurrentX to 0
    CharWidth = Picture1.TextWidth(Chr$ (KeyAscii))
    Picture1.CurrentX = CursorX + CharWidth: Picture1.CurrentY = Y1
    tBuf(R)=Fix(Picture1.CurrentX)
    Label1.Move Picture1.CurrentX, Y1, 30, Y2
    Timer1.Enabled = True
End Sub
```

行距示例使用了一个标题控制对象来模拟一个文本光标。使用 Line 语句来画一个文本光标也是可能的，不过必须考虑到执行 Line 语句将会改变 CurrentX 和 CurrentY 的值。下面的代码示例说明了使用 Line 语句绘制文本光标可能采用的方法。

下面是 Line 语句画光标的例子：

```
CursorX = Picture1.CurrentX          'save CurrentX
CharWidth = Picture1.TextWidth(Chr$ (KeyAscii))
Picture1.Line(CursorX. ,Y1-(CursorX+CharWidth,Y1+Y2),Picture1.BackColor,BF
Picture1.CurrentX=CursorX:Picture1.CurrentY=Y1
Picture1.Print Chr$ (KeyAscii) 'forces CurrentX to 0
Picture1.CurrentX=CursorX+CharWidth:Picture1.CurrentY=Y1
```

前面讲过的示例其功能是有限的，并且没有考虑存储或编辑被显示的文本。你也许希望在当地图书馆中找到有关计算机字处理的资料。实际上从 PC 机出现的那一天开始就可以编辑显示在计算机屏幕上的文本。我在图书馆中见到的一些比较老的计算机书籍就已经有足够的信息使你步入正轨了。

第 8 章　API 绘图程序

图 8.1 中的 API 绘图函数窗体可用来访问最常用的一些 Windows API 图形语言调用。16 位 Windows 操作系统的三个主要的动态链接库(.DLL)中，有选择地包含了功能强大的应用程序接口(API)。在 Windows 3.11 中，这几个文件位于 Windows 系统目录并被称作 GDI.EXE，USER.EXE 和 KRNL386(286).EXE。其中包含了 1000 多个可用于编程的 API 函数。Microsoft SDK(软件开发工具包)中包含了关于 API 的完整的参考资料，通过 CD 或调制解调器可从微软得到微软知识库(Microsoft Knowledge Base)，其中有 API 的补充资料。

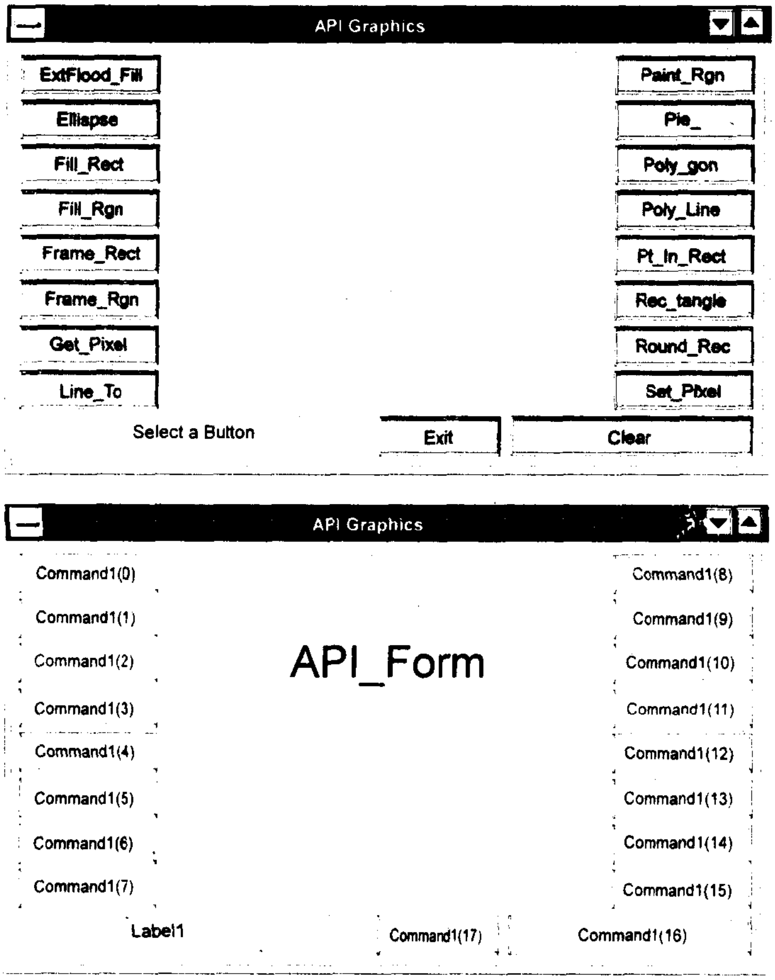

API Graphics
ExtFlood_Fill
Ellipse
Fill_Rect
Fill_Rgn
Frame_Rect
Frame_Rgn
Get_Pixel
Line_To
Paint_Rgn
Pie_
Poly_gon
Poly_Line
Pt_In_Rect
Rec_tangle
Round_Rec
Set_Pixel
Select a Button
Exit
Clear
API Graphics
Command1(0)
Command1(1)
Command1(2)
Command1(3)
Command1(4)
Command1(5)
Command1(6)
Command1(7)
API_Form
Command1(8)
Command1(9)
Command1(10)
Command1(11)
Command1(12)
Command1(13)
Command1(14)
Command1(15)
Label1
Command1(17)
Command1(16)

图 8.1　API 窗体及其控制对象

32 位 Windows 操作系统使用 GDI32 文件来处理 32 位(Long)API 函数。为了把许多 16 位 API 函数转换为 32 位，通常要把它重新声明为 32 位(Long)值，一些传给 32 位函数的参数值也必须以 Long(&)类型传过去。

下面是 16 位声明的一个例子：

```
Declare Function Polyline Lib "GDI" (ByVal hDC As Integer,
    lpPoints As POINTAPI,ByVal nCount As Integer) As Integer
```

下面是 32 位声明的例子。注意此例中的 hDC 和 nCount 参数是 Long(&)型。

```
Declare Function Polyline Lib "GDI32"(ByVal hDC As Long,
    lpPoints As POINT,ByVal nCount As Long) As Long
```

本章结尾处列出了最常用的一些 32 位 API 绘图函数。

如果窗体用到某函数,则应在该窗体 Declaration 部分声明一个 DLL 过程,或者在一个模块窗体中声明,以便程序中所有窗体均可使用此函数。下面是一个简单的 Declaration 举例。注意下面的声明必须输入连续的一行中。

```
Declare Function LineTo Lib "GDI" (ByVal hDC As Integer,
    ByVal X As Integer, ByVal Y As Integer) As Intger
```

可以用数据类型的后缀来简写一个函数的声明:

```
Declare Functon LineTo% Lib "GDI" (ByVal hDC%,ByVal X%,ByVal Y%)
```

Declare Function 或 Declare Sub:Visual Basic 命令。该命令说明正在声明一个函数或子程序。

LineTo%:本例中采用的 API 函数名。此处必须为库中的实际函数名。

Lib "User":函数所在的 DLL 库文件名。本域的另一可用选项是别名关键字。可参看《Visual Basic 编程指南》中的“调用 DLL 中的过程”一章。

API 函数声明的第二部分(在括号中)包括以下内容:

ByVal hDC%:带有焦点的窗口、控制对象或其他对象。许多函数都需要这个信息。

hDC 即设备环境图柄。设备环境通常指 Visual Basic 窗体或控制对象。比如,Artwork. hDC 通知 API 去寻找一个图柄,它标识了一个叫做 Artwork 的图片框窗口。API 将寻找包含了该图片框所有信息的设备环境即属性表。另外还有一些图柄标识符,如 hBITMAP(位图)和 hWnd(窗口)。

ByVal X%,ByVal Y%:这是传给函数的参数,包括值和选项设置的一个示例。ByVal 子句说明以值传送参数。除非是表达式或常数,否则 Visual Basic 均以引用方式传递参数。

某些参数的使用要求在模块窗体中加入用户定义的 Type 结构。从“Visual Basic 参考手册”中可得到更多细节,也可以从 Visual Basic 的文档里找到有关 API 函数的更多信息。Visual Basic 3.0 包含了 API 拷贝和粘贴工具。

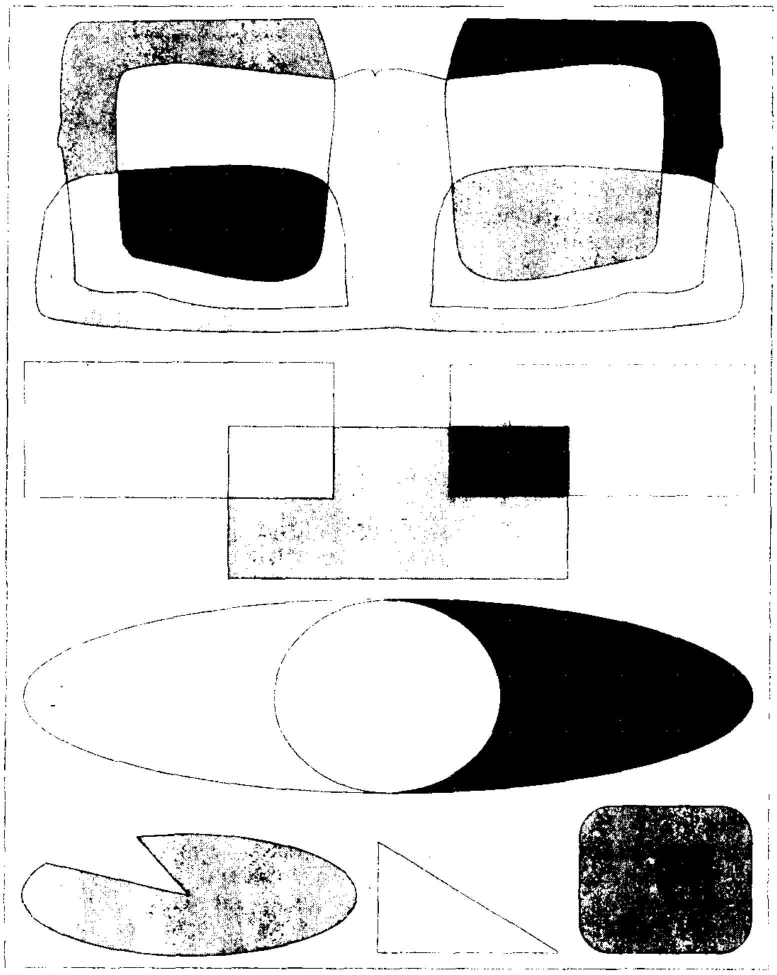
APIdraw 程序的窗体和控制对象的属性列于表 8.1。

表 8.1 APIdraw 窗体和控制对象的属性

Object: Form
Object Name:API_Form

BackColor	=&H00FFFFFF&	Left	=450
Caption	="API Graphics"	ScaleMode	=3 'Pixel
DrawMode	=6 'Invert	Top	=1305
Height	=4320	Width	=7935

```
Object:Command Button
Object Name:Command1 18 indexed controls
    Height        =375                      Width      =1245
    Caption       ="ExtFloodFill":Index0    Caption    ="Pie":Index 9
    Caption       ="Ellipse":Index 1        Caption    ="Polygon":Index 10
    Caption       ="FillRect":Index 2       Caption    ="Polyline":Index 11
    Caption       ="FillRgn":Index 3        Caption    ="PtIn_Rect":Index 12
    Caption       ="FrameRect":Index 4      Caption    ="Rectangle":Index 13
    Caption       ="FrameRgn":Index 5       Caption    ="RoundRect":Index 14
    Caption       ="GetPixel":Index 6       Caption    ="SetPixel":Index 15
    Caption       ="LineTo":Index 7         Caption    ="Clear Screen":Index 16
    Caption       ="PaintRgn":Index 8       Caption    ="Exit":Index 17
Object: Label
Object Name:Label1
    Alignment     =2 'Center                Left       =105
    BorderStyle   =1 'Fixed Single          Top        =3465
    Height        =330                      Width      =3690
```



8.1　APIdraw 程序的声明部分

下列声明是用于 16 位 Windows 操作系统的。你也许希望把 16 位 API 函数修改为将来可在 32 位 Windows 操作系统中使用的 GDI32 函数。

清单 8.1 API 程序使用的变量名

```
Declaration
Dim BoundingBox As Rectangle
Dim BoundingBox2 As Rectangle
Dim B As Integer
Dim W(1 To 20) As Integer
Dim N(1 To 20) As Integer
Dim E(1 To 20) As Integer
Dim S(1 To 20) As Integer
Declare Function ExtFloodfill% Lib "GDI" (ByVal hDC%,
    ByVal X%, ByVal Y%, ByVal crColor As Long, ByVal wFillType%)
Declare Function Ellipse% Lib "GDI" (ByVal hDC As Integer, ByVal X1 As Integer,
    ByVal Y1 As Integer, ByVal X2 As Integer, ByVal Y2 As Integer)
Declare Function FillRect% Lib "User" (ByVal hDC%, R As Rectangle,
    ByVal hBrush%)
Declare Function FillRgn% Lib "GDI" (ByVal hDC%, ByVal hrgn%, ByVal hBrush%)
Declare Function FrameRect% Lib "User" (ByVal hDC%, R As Rectangle,
    ByVal hBrush%)
Declare Function FrameRgn% Lib "GDI" (ByVal hDC%, ByVal hrgn%,
    ByVal hBrush%, ByVal nWidth%, ByVal nHeight%)
Declare Function GetPixel& Lib "GDI" (ByVal hDC%, ByVal X%, ByVal Y%)
Declare Function LineTo% Lib "GDI" (ByVal hDC As Integer, ByVal X As Integer,
    ByVal Y As Integer)
Declare Function PaintRgn% Lib "GDI" (ByVal hDC%, ByVal hrgn%)
Declare Function Pie% Lib "GDI" (ByVal hDC%, ByVal X1%, ByVal Y1%, ByVal X2%,
    ByVal Y2%, ByVal X3%, ByVal Y3%, ByVal X4%, ByVal Y4%)
Declare Function Polygon% Lib "GDI" (ByVal hDC%, P As PointXY, ByVal N%)
Declare Function Polyline% Lib "GDI" (ByVal hDC%, P As PointXY, ByVal N%)
Declare Function PtInRectBynum% Lib "User" Alias "PtInRect" (R As Rectangle,
    ByVal Pnt&)
Declare Function Rectangle Lib "GDI" (ByVal hDC As Integer, ByVal X1 As Integer,
    ByVal Y1 As Integer, ByVal X2 As Integer, ByVal Y2 As Integer) As Integer
Declare Function RoundRect Lib "GDI" (ByVal hDC As Integer, ByVal X1 As Integer,
    ByVal Y1 As Integer, ByVal X2 As Integer, ByVal Y2 As Integer,
    ByVal X3 As Integer, ByVal Y3 As Integer) As Integer
Declare Function SetPixel& Lib "GDI" (ByVal hDC%, ByVal X%, ByVal Y%,
    ByVal crColor&)
'''Other calls needed to preform above functions''''
Declare Function MoveTo Lib "GDI" (ByVal hDC As Integer, ByVal X As Integer,
    ByVal Y As Integer) As Long
Declare Function GetCurrentPosition& Lib "GDI" (ByVal hDC As Integer)
Declare Function CreateEllipticRgn% Lib "GDI" (ByVal X1%, ByVal Y1%,
    ByVal X2%, ByVal Y2%)
Declare Function CreateRectRgn% Lib "GDI" (ByVal X1%, ByVal Y1%, ByVal X2%,
    ByVal Y2%)
Declare Function CreateSolidBrush% Lib "GDI" (ByVal crColor&)
Declare Function DeleteObject% Lib "GDI" (ByVal hObject%)
Declare Function SelectObject% Lib "GDI" (ByVal hDC%, ByVal hObject%)
''''''''''''''''''''''''''''

Const INVERTMODE = 6
Const Left_Mouse = 1
Dim Toggle As Integer
Dim X1 As Single, Y1 As Single, X2 As Single, Y2 As Single
Dim CurrentButton As Integer
```

ArtAPI 程序用 Type Rect 数据结构来保存一个有界框的大小。在第 2 章“Artisan 程序”里对 Artisan 程序的介绍部分已讲过了 Type 变量结构。

Dim BoundingBox As Rectangle：API 函数 FrameRect 使用 Type Rect 结构来显示矩形的轮廓。APIDRAW.BAS 中的 Type Rect 结构定义了对象的左、上、右、下位置。BoundingBox 即用以保存这些位置。示例如下：

```
BoundingBox. Left=10:BoundingBox. Top=15
BoundingBox. Right=150:BoundingBox. Bottom=100
```

大多数 API 绘图函数都需要以象素为单位的尺寸或坐标。

Dim BoundingBox2 As Rectangle：用于本程序的 Pt_In_Rect 示例中。

Dim B As Integer：用来保存绘制的矩形个数。

Dim W(1 To 20) As Integer：保存被绘制对象的西边或者说左边坐标。N(index)，E(index)，S(index)分别保存北(上)、东(右)、南(下)坐标。W、N、E 和 S 数组中共可存放 20 组坐标值。

Declare Function ExtFloodfill%：本程序用到的几个 API 绘图函数之一。编程时可用的 API 函数超过 1000 个。

API 绘图函数

下表列出了最常用的 Windows 3.11 API 绘图函数。另外还有关于画笔、刷子、图元文件和位图的 API 函数。32 位 Windows 操作系统增强了画线和图元文件函数，同时使程序员可以产生 Bézier 曲线。

Arc
Chord
DrawFocusRect
Ellipse
ExtFloodFill
FillRect
FillRgn
FloodFill
FrameRect
FrameRgn
GetBkColor
GetBkMode
GetCurrentPosition
GetNearestColor
GetPixel
GetPolyFillMode
GetROP2
GetROP2
LineTo
MoveTo
MoveToEx
PaintRgn
Pie
PolyGon
PolyLine
PolyPolygon
Rectangle
RoundRect
SetBkColor
SetPixel
SetPoly
SetROP2

清单 8.2 列出了 Form_Load 事件启动时的默认值。

清单 8.2　Form_Load 事件启动时的默认值

```
Sub Form_Load ()
    Command1_Click (16)
    Toggle =True
End Sub
```

Command1_Click(16)：触发清屏按钮(索引号为 16)。

Toggle：用在 Poly_gon 和 Poly_line 过程中以切换每条新线段的起始、终点位置，即将 X1，Y1 切换为 X2，Y2，或相反。

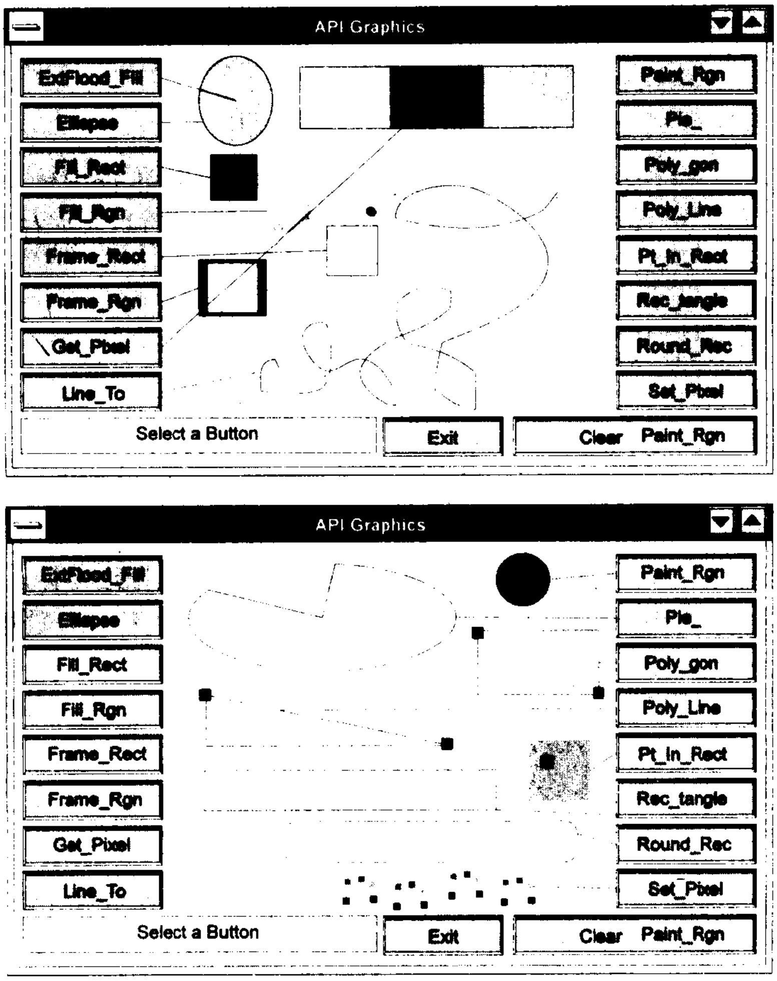

API Graphics
ExtFlood_Fill
Ellipse
Fill_Rect
Fill_Rgn
Frame_Rect
Frame_Rgn
Get_Pixel
Line_To
Paint_Rgn
Pie_
Poly_gon
Poly_Line
Pt_In_Rect
Rec_tangle
Round_Rec
Set_Pixel
Select a Button
Exit
Clear Paint_Rgn

图 8.2 可用绘图函数

8.2 单击一个 API 函数按钮

窗体中 17 个被索引的 Command1 按钮中的每一个分别执行一个 API 绘图函数，其中大多数函数只有在单击窗体时才会执行。当单击一按钮时，其索引号保存至变量 CurrentButton。窗体通过检查 CurrentButton 的值即可知道选择了哪个按钮。CommandButton_Click 事件的主要部分是显示 Label1 控制对象中的消息，并以此指导你下一步怎样做。

清单 8.3 显示对每个按钮应采取的动作

```
Sub Command1_Click (Index As Integer)
CurrentButton = Index

Select Case CurrentButton
    Case 0 'ExtFloodFill
        Label1.Caption = "Fill Area With Color"
    Case 1 'Ellipse
        Label1.Caption = "Drag to Draw Circle"
    Case 2 To 5 'Fill Rgn To Frame Rgn
        Label1.Caption = "Mouse Down"
        Case 6 'Get Pixel
            Label1.Caption = "Click Color Palette"
            Cls: DrawMode = 13
            Line (180, 50)-(230, 100), QBColor(1), BF
            Line (230, 50)-(280, 100), QBColor(2), BF
            Line (280, 50)-(330, 100), QBColor(3), BF
            DrawMode = 6
        Case 7 'Line_To
            Label1.Caption = "Drag to Draw Lines"
        Case 8 'Paint_Rgn
            Label1.Caption = "Mouse Down"
        Case 9 'Pie_
            Label1.Caption = "Drag to Draw Pie Shape"
        Case 10 'Poly_gon
            Label1.Caption = "Pick two Corner Points"
        Case 11 'Poly_Line
            Label1.Caption = "Pick two Angle Points"
        Case 12 'Pt_In_Rect
            Label1.Caption = "Draw Some Rectangles First"
        Case 13 To 14 'Rec_Tangle to Set_Pixel
            Label1.Caption = "Drag Mouse"
        Case 15 'Rec_Tangle to Set_Pixel
            Label1.Caption = "Drag Mouse"
            Line (0, 0)-(ScaleWidth, ScaleHeight \ 2), , BF
        Case 16 'CLS
            Label1.Caption = "Select a Button": Cls
        Case 17
            End
    End Select
    End Sub
```

CurrentButton：保留被选择按钮的索引号。

Select Case：根据所选按钮，执行后续 Case 表达式之一。

Case 0：选择了标题为 Flood_Fill 的按钮。

Case 1 To 17：同 Case 0 一样，选择了以 API 函数名为标题的按钮。大多数情况下，标题控制对象中会显示一条信息来告诉你下一步怎样做。在 Case 6 中，为了更好地说明 GetPixel 而画了一个三块式的调色板。Case 15 中，画了一个矩形以加强 SetPixel API 函数示例。

API 画笔和刷子函数

下表列出了最常用的 Windows 3.11 API 画笔和刷子函数。

CreatHatchBrush
SelectObject
GetBrushOrg
CreatBrushIndirect
CreatePenIndirect
CreatPatternBrush
SetBrushOrg
GetStockObject
DeleteObject
CreateSolidBrush
CreatePen
UnrealizeObject

API 矩形和区域函数(边界框)

下表列出了一些最常用的 Windows 3.11 API Rect 和 Rgn 函数。

- CopyRect
- CreateRectRgn
- EqualRect
- FillRect
- FillRgn
- FrameRect
- FrameRgn
- GetBoundsRect
- GetRgnBox
- InfateRect
- OffsetRect
- Rectangle
- SetRect
- SetRectRgn
- SetRectEmpty
- SubtractRect

8.3 窗体的鼠标事件

清单 8.4 中的 Form_MouseDown 事件使用 Type Rect 结构创建了可用于 API 函数的具有默认尺寸的矩形。Select Case 语句使用 CurrentButton 的值来转入到含有正确过程的 Case 块。CurrentButton 的值即为所选按钮的索引。

清单 8.4 设定第一个动作或过程调用

```
Sub Form_MouseDown (Button%, Shift%, X As Single, Y As Single)
If CurrentButton = 2 Or CurrentButton = 4 Then 'needed for FillRect or FrameRect
    BoundingBox.Left = X - 20
    BoundingBox.Top = Y - 20
    BoundingBox.Right = X + 20
    BoundingBox.Bottom = Y + 20
End If
Select Case CurrentButton
    Case 2 'FillRect
        hndBrush% = CreateSolidBrush(RGB(0, 0, 255))
        R = FillRect(API_Form.hDC, BoundingBox, hndBrush%)
        R = DeleteObject(hndBrush%)
    Case 3 'FillRgn
        DrawMode = 13
        hrgn% = CreateRectRgn(X - 20, Y - 20, X + 20, Y + 20)
        hndBrush% = CreateSolidBrush(RGB(255, 0, 255))
        R = FillRgn(API_Form.hDC, hrgn%, hndBrush%)
        R = DeleteObject(hndBrush%)
        R = DeleteObject(hrgn%)
    Case 4 'FrameRect
        hndBrush% = CreateSolidBrush(RGB(0, 0, 255))
        R = FrameRect(API_Form.hDC, BoundingBox, hndBrush%)
        R = DeleteObject(hndBrush%)
    Case 5 'FrameRgn
        DrawMode = 13
        hrgn% = CreateRectRgn(X - 20, Y - 20, X + 20, Y + 20)
        hndBrush% = CreateSolidBrush(RGB(255, 0, 0))
        nWidth% = 8: nHeight% = 2
        R = FrameRgn(API_Form.hDC, hrgn%, hndBrush%, nWidth%, nHeight%)
        R = DeleteObject(hndBrush%)
        R = DeleteObject(hrgn%)
    Case 6
        Get_Pixel Button, Shift, X, Y
    Case 8 'PaintRgn
        hrgn% = CreateEllipticRgn(X - 20, Y - 20, X + 20, Y + 20)
```

```
          R = PaintRgn(API_Form.hDC, hrgn%)
          R = DeleteObject(hrgn%)
      Case 10 'Polygon
          DrawWidth = 4: PSet (X, Y): DrawWidth = 1
          Poly_gon Button, Shift, X, Y: Exit Sub
      Case 11 'Polyline
          DrawWidth = 4: PSet (X, Y): DrawWidth = 1
          Poly_Line Button, Shift, X, Y: Exit Sub
      Case 12 'Pt In Rect
          Pt_In_Rect Button, Shift, X, Y: Exit Sub
  End Select
          X1 = X: Y1 = Y: X2 = X: Y2 = Y
          R = MoveTo(API_Form.hDC, X, Y)
  End Sub
```

If CurrentButton：如果选择了 Fill_Rect 或 Frame_Rect 按钮，则要用一 Type 数据结构定义矩形尺寸。

BoundingBox. Left(Top,Right,Bottom)：DRAWAPI. BAS 文件中定义的 Type 结构。

CurrentButton：记录着 17 个索引按钮中被选择的按钮。

Select Case：根据所选按钮，执行后续 Case 表达式：

Case 2：API 函数 FillRect。

CreateSolidBrush：创建填充矩形用的颜色。

FillRect：API 函数 FillRect 需要以下参数：1. 在哪个控制对象中绘制；2. 四个屏幕坐标点；3. 填充用的颜色。

DeleteObject：删去填充颜色刷子，以便可用于其他事件。

Case 3：API 函数 FillRgn。

CreateRectRgn：手工产生矩形的 4 个屏幕点。

CreateSolidBrush：创建填充矩形用的颜色。

FillRgn：API 函数 FillRgn 需要以下参数：1. 在哪个控制对象中绘制；2. 4 个屏幕坐标点；3. 填充用的颜色。

DeleteObject：操作完毕即删去填充颜色和定制矩形。

Case 4：API 函数 FrameRect。

CreateSolidBrush：创建矩形边框颜色。

FrameRect：API 函数 FrameRect 需要以下参数：

1. 在哪个控制对象中绘制；2. 4 个屏幕坐标点；3. 填充用的颜色。

DeleteObject：删去填充用颜色刷子，以便可用于其他事件。

Case 5：API 函数 FrameRgn。

CreateRectRgn：手工产生矩形的 4 个屏幕点。

CreateSolidBrush：创建矩形边框颜色。

nWidth：等于侧边边框宽度。

nHeight：等于上下边框宽度。

FrameRgn：API 函数 FrameRgn 需要以下参数：

1. 在哪个控制对象中绘制；2. 4 个屏幕坐标点；3. 边框颜色；4. 侧边框宽度；5.

上下边框宽度。

DeleteObject：操作完毕即删去填充颜色和定制矩形。

Case 6：直接转向 Get_Pixel 过程。

Case 8：API 函数 PaintRgn。

CreateEllipticRgn：手工产生椭圆的 4 个点。

PaintRgn：在哪个控制对象中绘制，以及要画的结构化对象。

DeleteObject：删去定制对象，以便可用于其他事件。

Case 10：直接转向 Poly_Gon 过程。

Case 11：直接转向 Poly_Line 过程。

Case 12：直接转向 Pt_In_Rect 过程。

最后，将鼠标的当前 X、Y 位置赋给其他变量，以便通过窗体的 MouseMove 事件得到一个可拉伸的边界框。

X1，Y1，X2，Y2：用来绘制对象的标准 MouseDown 例程。

R＝MoveTo(API_Form. hDC，X，Y)：API 函数 MoveTo 用来指示绘制自由线条时的起点坐标，并被用于 Line_To 过程。

清单 8.5 中的 Form_MouseMove 过程在执行时可分支到多个子程序，从而可利用鼠标的 X 和 Y 坐标来绘制移动对象。

清单 8.5　API 绘图操作

```
Sub Form_MouseMove (Button%, Shift%, X As Single, Y As Single)
If Button = Left_Mouse Then
Select Case CurrentButton
    Case 1    'Ellipse
        Ellipse_ Button, Shift, X, Y
    Case 7    'Line To
        Line_To Button, Shift, X, Y
    Case 9    'Pie
        Pie_ Button, Shift, X, Y
    Case 13 'Rectangle
        Rec_tangle Button, Shift, X, Y
    Case 14 'RoundRec
        Round_Rect Button, Shift, X, Y
    Case 15 'Set Pixel
        Set_Pixel Button, Shift, X, Y
End Select
End If
End Sub
```

If Button＝Left_Mouse Then：如果鼠标左键被按下，则继续。

Select Case CurrentButton：Select Case 语句根据 CurrentButton 的值分支到适当的 Case 块。CurrentButton 的值即为所选按钮的索引号。

Case 1 To 15：分支到相应过程。所有过程的参数均为鼠标按钮、换档键和鼠标移动时的

X 和 Y 坐标。

8.4 用 FloodFill 函数绘图

当选择了 FloodFill 按钮时，清单 8.6 中的 Form_MouseUp 事件将以随机的颜色填满整个窗体。MousUp 事件同时也计算已画在窗体中的矩形个数。

清单 8.6 设置或重置有用的变量值

```
Sub Form_MouseUp (Button%, Shift%, X As Single, Y As Single)
Select Case CurrentButton
    Case 0 ' FloodFill
        DrawMode = 13
        FillStyle = 0
        color% = (14 * Rnd) + 1
        FillColor = QBColor(color%)
         R = ExtFloodfill(API_Form.hDC, X, Y, &H8, FloodFillBorder)
    Case 13 ' Rectangle
        If B < 20 Then B = B + 1
        W(B) = X1: N(B) = Y1: E(B) = X2: S(B) = Y2
End Select
DrawMode = 6: FillStyle = 1
End Sub
```

Select Case CurrentButton：Select Case 语句根据 CurrentButton 的值分支到正确的 Case 块。CurrentButton 是所选按钮的索引号。

Case 0：API 函数 FloodFill。

DrawMode=13:FillStyle=0：绘制方式设为 13，即正常方式(copy_pen)。为了可以观察到 FloodFill 所用的颜色，必须把默认填充方式从透明(1)改为实心(0)，或某一种图案填充方式(2 到 7)。

FillColor：从 QBasic 的 16 中颜色中，随机选择除黑色(0)以外的一种颜色。此颜色用来画窗体的背景。

R=ExtFloodfill：API 函数 ExtFloodfill 需要以下参数：1. 在哪个控制对象中绘制；2. 开始颜色填充的 X,Y 坐标；3. 边框颜色；4. 所用填充方式。

Case 13：保存用于 Pt_In_Rect 函数的矩形的坐标。最多可画 20 个矩形。

8.5 API 绘图函数

调用 API 过程时的返回值变量是一虚设变量，可以起任何满足 Visual Basic 命名标准的名字，例如，API，Dummy 或 R 等。很多情况下此虚设变量的值可表示是否成功地调用了 API 过程。如：

```
API=LineTo(API_Form.hDC,X,Y)
```

如果 LineTo 函数执行成功，API 变量将返回一非零值(如－1)，如果失败返回零(0)。

有时成功的 API 调用返回指定的窗口(图柄)个数或其他有用信息。可以在调用 API 函

数后马上使用 ADD Watch,debug,print 或 Format $ (虚变量名)等 Visual Basic 语句来观察返回值。如:

```
Print dummy name,Label1.Caption=Format $ (dummy name)
```

下面讲述包含 API 绘图函数的过程。

8.6 椭圆函数(Ellipse)

可用 Ellipse 函数绘制一个圆或椭圆。该函数见清单 8.7。

清单 8.7 绘制可变圆或椭圆的子例程

```
Sub Ellipse_(Button As Integer, Shift As Integer, X As Single, Y As Single)
R = Ellipse%(API_Form.hDC, X1, Y1, X2, Y2)
     X2 = X
     Y2 = Y
R = Ellipse%(API_Form.hDC, X1, Y1, X2, Y2)
End Sub
```

Ellipse%:Ellipse%函数需要知道在哪个控制对象里绘图(API_Form.hDC)和椭圆的 4 个屏幕坐标点(X1,Y1,X2 和 Y2)。此圆或椭圆就位于 X1,Y1,X2 和 Y2 构成的边界框中。

X2=X;Y2=Y:使边界框的另一角位于当前鼠标处。现在使用的是反显绘图模式,头一次绘制使用了背景的反转色(从白到黑)。当下一次在黑色象素上绘图时,就变为白色(反之亦同)。这就产生了移动的效果。

Ellipse%:为产生移动效果,再次调用椭圆函数。

8.7 读象素函数(Get_Pixel)

Get_Pixel 函数返回一象素点颜色值,这是一个十六进制值。清单 8.8 列出了 Get_Pixel 函数程序,它调用了 API 函数 GetPixel。

清单 8.8 Get_Pixel 返回一象素的红、绿、蓝值

```
Sub Get_Pixel (Button As Integer, Shift As Integer, X As Single, Y As Single)
    RGB_Value = GetPixel(API_Form.hDC, X, Y)
    Msg $  = "RGB Value = "
    Label1.Caption = Msg $ + RGB_Value
End Sub
```

RGB_Value:被单击象素的红、绿、蓝三色值。API 函数 GetPixel 需要以下参数:1. 在哪个控制对象中单击鼠标;2. 鼠标位置。若执行成功,则返回一个十六进制值。

Label1.Caption:显示于 Label1 中的说明,如“RGB Value=255”。

8.8 画线函数(Line_To)

Line_To 函数类似于 Basic 中的 PSet 和 Step Line 语句,也类似于 Postscript 打印机语言中跳到当前页中某指定坐标的语句。

清单 8.9　用 Line_To 过程画连续线

```
Sub Line_To (Button As Integer, Shift As Integer, X As Single, Y As Single)
    R = LineTo(API_Form.hDC, X, Y)
End Sub
```

LineTo(hDC,X,Y)：API 函数 LineTo 需要知道在哪个控制对象中(API_Form.hDC)开始或移动坐标。当按下鼠标按钮并移动鼠标时，本过程将画出一随意线条。

8.9　API 饼图函数(Pie)

清单 8.10 中的 Pie 过程使用了 Pie_函数。Pie 函数非常不适合画可变饼图。因为当鼠标移动时，X3，Y3，X4，Y4 也发生改变，饼图边上的楔形缺口就会偏离象素的交点。对于 RoundRect 函数中的 X3，Y3 也是这样。

为了正确显示饼图形状，其楔形缺口应在饼图变化时保持为常值。请参考随书磁盘的 16MISC 目录中的 MetaFile 程序，可以找到具有常数缺口值的饼图的示例。

我建议使用标准的 Visual Basic Circle 语句来显示可变饼图。在第 11 章“绘制圆角程序”中，有移动由直线段和曲线段构成的对象的示例。

饼图的楔形缺口是在圆弧上另外画的。楔形的两边是连往圆心的直线，这两条线的端点决定了它们的角度。

在清单 8.10 的 Pie 函数中，一条边随着鼠标移动，另一条边则随机绘制。请慢点移动鼠标，以观察饼图的变化。

清单 8.10　绘制可变饼图

```
Sub Pie_(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ForeColor = BackColor
    R = Pie(API_Form.hDC, X1, Y1, X2, Y2, X3, Y3, X4, Y4)
       X2 = X
       Y2 = Y
       X3 = X2                  'wedge side follow mouse
       Y3 = Y2
       X4 = Int(X2 * Rnd) 'wedge side is random
       Y4 = Int(Y2 * Rnd)
    BackColor = ForeColor
    R = Pie(API_Form.hDC, X1, Y1, X2, Y2, X3, Y3, X4, Y4)
End Sub
```

ForeColor＝BackColor：由于 API 函数 Pie 严重影响绘图模式，故反转此动作。不过切换 ForeColor 和 BackColor 的作法并不值得推荐，并且你也许希望人工绘制饼图。请参考第 11 章“绘制圆角程序”。

Pie：Pie 函数需要知道在哪个控制对象中绘图(API_Form.hDC)以及构成饼图形状的 8 个点(X1，Y1，X2，Y2，X3，Y3，X4 和 Y4)。

X2＝X：Y2＝Y：使边界框的一角位于当前鼠标位置，这样会产生移动效果。

X3，Y3：随着鼠标移动的一条楔形缺口边。

X4，Y4：另一条边，位于随机位置。

BackColor＝ForeColor：再次反转绘图颜色。

Pie：为产生移动效果，再次调用 Pie 函数。

8.10　多边形函数(Polygons)

多边形是由多条线段首尾相接构成的闭合图形。这里提到了“闭合”多边形，三角形就是一个具有三个点的多边形。清单 8.11 中的 Polygon 函数可用来画多边形。

清单 8.11　绘制多边形

```
Sub Poly_gon (Button As Integer, Shift As Integer, X As Single, Y As Single)
Dim N As Integer
Static POINTS(1 To 5) As PointXY
    If Toggle = True Then
        X1 = X: Y1 = Y
    Else
        X2 = X: Y2 = Y
POINTS(1).X = X1 'Draw a Star shape
POINTS(1).Y = Y1
POINTS(2).X = X1 - (X2 - X1)
POINTS(2).Y = Y2
POINTS(3).X = X2
POINTS(3).Y = Y1 + ((Y2 - Y1) \ 3)
POINTS(4).X = X1 - (X2 - X1)
POINTS(4).Y = Y1 + ((Y2 - Y1) \ 3)
POINTS(5).X = X2
POINTS(5).Y = Y2
N = 5
R = Polygon(API_Form.hDC, POINTS(1), N)
End If
Toggle = Not Toggle
End Sub
```

Dim N As Integer：N 代表多边形中的端点的总数。

Static POINTS(1 To 5)As PointXY：POINTS 为存放多边形端点在屏幕上位置的数组。X 和 Y 坐标均以象素为单位。X，Y 屏幕坐标点用 APIDRAW.BAS 文件中的一个 Type 结构来定义。本例中使用了五组 X，Y 值(1 To 5)。如要画具有更多端点的多边形，可增大此数组。

If Toggle=True Then：在第一次单击鼠标按钮时，Toggle 等于 True，这样 X1 和 Y1 就等于当前鼠标位置。第二次单击鼠标时，Toggle 等于 False，并且设置 POINT 变量 2，3，4 和 5 的 X，Y 值，从而画出一多边星形。

POINT(1).X；POINT(1).Y：多边形第一点的 X 和 Y 值。

POINT(2).X；POINT(2).Y：多边形第二点的 X 和 Y 值。

POINT(3).X；POINT(3).Y：多边形第三点的 X 和 Y 值。

POINT(4).X；POINT(4).Y：多边形第四点的 X 和 Y 值。

POINT(5).X；POINT(5).Y：多边形第五点的 X 和 Y 值。

N=5：本次绘图用五个点的坐标。如果多边形有更多端点，则相应增加这个数目。

R=Polygon：本函数需要以下参数：1. 在哪个控制对象中绘图(.hDC)；2. 绘图起始点

(POINTS(1));3. X,Y 坐标个数。例如：

```
API=Polygon(API_Form.hDC,POINTS(1),N)
```

多边形的第一点为指定索引号的数组值(Point(index)),然后用更高索引号的数组值来绘制,直到索引号为 N。如果起始点的索引号大于数组的大小,将发生错误。

Toggle=Not Toggle:Toggle 变量初值为 True。Not 操作使 True 变为 False。下一次执行此语句时,False 变为 True。

8.11　折线函数(Polyline)

API 函数 Polyline 与 Polygon 函数比较类似,不同的是起始点和终止点并不自动地连接,所以被归入“开”多边形。Polyline 函数与 Visual Basic 中使用 Step 选项的 Line 语句相同,不过有时侯用 Line 语句更容易些,因为可以少用些代码。清单 8.12 列出了画折线的程序。

清单 8.12　画折线

```
Sub Poly_Line (Button As Integer, Shift As Integer, X As Single, Y As Single)
Dim N As Integer
Static POINTS(1 To 5) As PointXY
    If Toggle = True Then
        X1 = X: Y1 = Y
    Else
        X2 = X: Y2 = Y
POINTS(1).X = X1 'Draw a Star shape
POINTS(1).Y = Y1
POINTS(2).X = X1 - (X2 - X1)
POINTS(2).Y = Y2
POINTS(3).X = X2
POINTS(3).Y = Y1 + ((Y2 - Y1) \ 3)
POINTS(4).X = X1 - (X2 - X1)
POINTS(4).Y = Y1 + ((Y2 - Y1) \ 3)
POINTS(5).X = X2
POINTS(5).Y = Y2
FirstPoint = 1: TotalPoints = 5
R = Polyline(API_Form.hDC, POINTS(FirstPoint), TotalPoints)
End If
Toggle = Not Toggle
End Sub
```

R=Polyline:除了始点和终点不自动连接外,Polyline 函数和 Polygon 函数基本一样。请参看清单 8.11,画折线的方法是类似的。

8.12　Pt_In_Rect 函数

清单 8、13 中的 PtInRect 例程对绘图程序来讲非常重要。API 函数 PtInRect 可以自动搜索一组边界框并判断鼠标是否位于某一个框中。

Artisan 程序将所画对象的边界框位置存储于分离的数组里(W(B),E(B),N(B),S(B)),然后搜索这四个数组以找到对象。也可以使用单一的 Type 结构或多重数组来保存每个边界框,然后调用 API 函数 PtInRect 来判定鼠标是否位于某个边界框或 Rect 结构中。

Artisan 程序正是用 API 函数 PtInRect 来找出鼠标所在的对象的。

清单 8.13　判定鼠标是否位于某边界框中

```
Sub Pt_In_Rect (Button As Integer, Shift As Integer, X As Single, Y As Single)
LastB = B
For B = 1 To 20
    BoundingBox2.Left = W(B)
    BoundingBox2.Top = N(B)
    BoundingBox2.Right = E(B)
    BoundingBox2.Bottom = S(B)
    XY& = X + CLng(Y) * &H10000
        If PtInRectBynum%(BoundingBox2, XY&) Then
            DrawWidth = 4: PSet (X, Y): Exit For
        End If
Next B
B = LastB
    If DrawWidth = 4 Then
        Label1.Caption = "Point is in Rectangle": DrawWidth = 1
    Else
        Label1.Caption = "Point is not in Rectangle"
    End If
End Sub
```

For B=1 To 20：本程序中最多可画 20 个矩形。For...Next 语句循环 20 次以使 PtInRect 函数可检查所有可能的边界框坐标。

BoundingBox.Left(Top,Right,Bottom)：这是定义在 ARTAPI.BAS 文件中的 Type 结构。PtInRect 函数要求以 Rect 结构传送矩形的左、上、右、下坐标。

XY&=X+CLng(Y)*&H10000：应记住此公式，用来以一个 LONG 值传送 X,Y 坐标。此公式把 16 位的 X 和 Y 组合在一起形成一个 32 位长值，X 在低位，Y 在高位。具体做法是把 Y 转换到长值(CLng)然后乘以 65536，再加上 X 即等于 XY&。

PtInRectBynum%：API 函数 PtInRectBynum 要求以下参数：1. 正在查找的边界框(BoundingBox2)；2. 落于此框中的 X,Y 点(XY&)。

If PtInRectBynum%：当单击一鼠标按下事件且鼠标位于边界框中时，则 PSet 语句画出一个大点来指示已找到边界框，Label.Caption 显示“Point is inside rectangle(点在矩形中)”。

8.13 矩形函数(Rec_tangle)

清单 8.14 中的 Rec_tangle 函数与 Visual Basic 中带 B 选项的 Line 语句类似，B 选项使 Line 语句画出一矩形。

清单 8.14　绘制可变矩形框

```
Sub Rec_tangle (Button As Integer, Shift As Integer, X As Single, Y As Single)
    R = Rectangle(API_Form.hDC, X1, Y1, X2, Y2)
      X2 = X
      Y2 = Y
    R = Rectangle(API_Form.hDC， X1， Y1， X2， Y2)
End Sub
```

R=Rectangle：API 函数 Rectangle 需要知道在哪个控制对象中绘图(API_Form.hDC)以及矩形的 4 个屏幕坐标点(X1,Y1,X2,Y2)。

X2=X:Y2=Y：把边界框的一角置于当前鼠标位置，以产生移动效果。

R=Rectangle：为产生变动效果，再次调用 Rectangle。

8.14　圆角矩形函数(Round_Rect)

清单 8.15 中的 Round_Rect 过程绘制带圆角的可变矩形。API 函数 Round_Rect 使用一个矩形和四段椭圆弧来构成一圆角矩形。出于同 Pie 过程(清单 8.10)类似的原因，此处也必须设 ForeColor 等于 BackColor。

清单 8.15　绘制可变圆角矩形

```
Sub Round_Rect (Button As Integer， Shift As Integer， X As Single， Y As Single)
ForeColor =BackColor
    R = RoundRect(API_Form.hDC， X1， Y1， X2， Y2， X3， Y3)
       X2 = X
       Y2 = Y
       X3 = 50
       Y3 = 50
       ForeColor =BackColor
    R = RoundRect(API_Form.hDC， X1， Y1， X2， Y2， X3， Y3)
End Sub
```

R=RoundRect：API 函数 Round_Rect 需要知道在哪个控制对象中绘图(API_Form.hDC)，以及 6 个屏幕坐标点(X1,Y1,X2,Y2,X3,Y3)。

X2=X:Y2=Y：把边界框的一角置于当前鼠标位置，以产生移动效果。

X3：椭圆弧宽(对任一圆角)。

Y3：椭圆弧高(对任一圆角)。

R=RoundRect：为产生变动效果，再次调用 Round_Rect。

8.15　设置象素函数(Set_Pixel)

清单 8.16 中的 Set_Pixel 过程与 Visual Basic PSet 语句基本相同。

清单 8.16　喷涂象素

```
Sub Set_Pixel (Button As Integer， Shift As Integer， X As Single， Y As Single)
    DrawMode = 13
    crColor& = QBColor(15 * Rnd)
    R=SetPixel(API_Form,hDC,X,Y,crColor&)
End Sub
```

DrawMode＝13：设置象素绘图模式为正常模式(Copy_Pen)。

crColor&：象素颜色值。

SetPixel：API 函数 SetPixel 需要以下参数：1. 在哪个控制对象中绘制(API_Form.hDC)；2. 象素位置(X 和 Y)；3. 象素颜色(crColor&)。

32 位 API 绘图函数

下面是一些常用的 32 位 API 函数，用于可定义 32 位 API 函数的 Windows 操作系统。

- AngleArc *
- AnimatePalette
- Arc
- ArcTo *
- BeginPath *
- BitBlt
- Chord
- CloseFigure *
- CopyEnhMetaFile *
- CreatBrushIndirect
- CreateDIBitmap
- CreateDIBPatternBrush
- CreateEnhMetaFile
- CreateFontIndirect
- CreatePatternBrush
- DeleteObject
- Ellipse
- EndPath *
- EnumEnhMetaFile *
- ExcludeClipRect
- ExtTextOut
- FillPath *
- FillRect
- FillRgn
- FloodFill
- GetDIBits
- Getpath *
- IntersectClipRect
- LineTo
- MakePath *
- MaskBlt *
- MoveToEx
- OffsetClipRgn
- OffsetViewportOrg
- OffsetWindowOrg
- PatBlt
- PathToRegion *
- Pie
- PlayEnhMetaFile8
- PlgBlt *
- PolyBézier *
- PolyBézierTo *
- PolyDraw *
- Polygon
- Polyline
- PolyPolygon
- PolyPolyline
- Realizepalette
- Rectangle
- RestoreDC
- RoundRect
- PtInRegion
- SaveDC
- ScaleViewportExt
- ScaleWindowExt
- SelectClipPath *
- SelectClipRgn
- SelectFont
- SelectObject
- SelectPalette
- SetArcDirection *
- SetBkColor
- SetMapMode
- SetWindowExtEx
- SetViewportExtEx
- SetViewportOrgEx
- SetBkMode
- SetDIBitsToDevice
- SetPixel
- SetPolyFillMode
- SetROP2
- SetStretchBltMode
- SetTextColor
- SetTextJustification
- SetViewportExt
- SetWindowExt
- SetWindowOrg
- StrokeAndFillPath *
- StrokeAndFillPath *
- StrokePath *
- StretchDIBits
- TextOut
- WidenPath *

* Windows API 库中的新函数。

8.16 本章小结

API 绘图函数可以替代 Visual Basic 的 Line、Circle 和 PSet 等语句，不过有时 Visual Basic 图形语句可以用少量代码完成 API 函数的功能。同时，有些图形操作 Visual Basic 语句则不易完成，因为无法用 Visual Basic 的方法、函数或语句来描述它们。比如 DOS 下 Basic 语言的 Paint 语句，其作用是填充屏幕上一块区域，在 Visual Basic 中，只有用 API 函数 FloodFill 或 ExtFloodFill 来完成同样功能。

第 15 章中的 ArtAPI 程序是改写第 2 章中的 Artisan 程序的结果。Artisan 程序使用基本的图形语句来绘制对象形状，ArtAPI 使用 API 函数来达到相同目的。当画一条线、一个框或圆时，可以不用 API 函数 LineTo、Rectangle 或 Ellipse，因为 Visual Basic 的 Line 和 Circle 语句就可以很好地完成这些工作。不过，在填充一个复杂形状时，仍然要用 API 图形函数。第 13 章“节点连接程序”说明了怎样用 API 图形函数来填充一个多边形对象图形。

第 9 章　填色与调色

喷涂填色又被称为梯度填色(gradient fills)。喷涂填色是指一个边界框被从一种色调连续地变化成另一种色调的颜色阶梯所填充。这种填色方法常用来表示彩虹或阴影的效果。调色板程序(Palette program)中使用竖直颜色带来实现喷涂填色。每一条颜色带都是通过 Line 语句中 box fill 的选项来实现的。为了在一个三角形内实现喷涂填色，就需要画好三角形框的每一条边并且在其中填色。在 DOS 版的 Basic 语言中，Draw 语句有一个角度选项，可以相对容易地实现它。然而，在 Visual Basic 中，就需要使用 API 函数中的 polyline 或 line 的画线方法及 FloodFill 函数进行填充。

Palette 程序给出了两种方法来实现从一种颜色到另一种颜色的渐变调色。图 9.1 给出了 Palette 程序的屏幕显示。单击以 Fountain Fill 为标题的 Fill_Button 控制对象，即可在 PicBlends 图形控制对象框内显示出每一层次的颜色，每一条颜色带均采用 Visual Basic 的 Line 语句来画框并填充颜色，Line 语句中的颜色项选用是通过 Palette 程序中的 RGBColor& 函数来安排的，正是通过这一函数来实现喷涂填色的颜色阶梯效果(见图 9.2)。

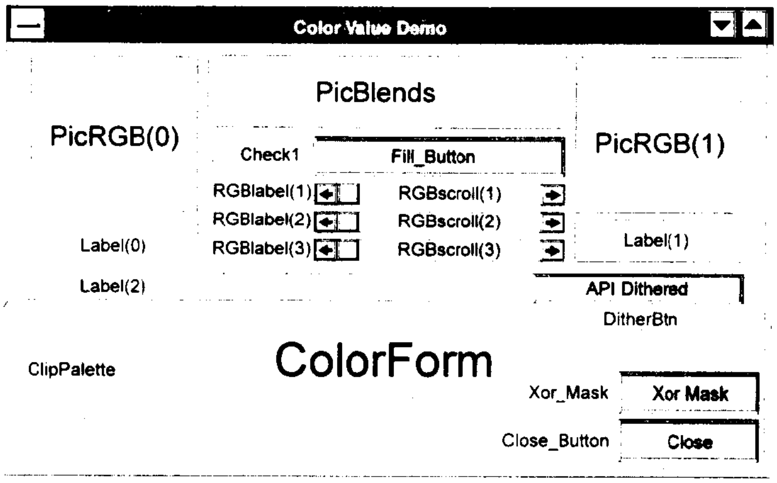

Color Value Demo
PicBlends
PicRGB(0)
Check1
Fill_Button
PicRGB(1)
RGBlabel(1)
RGBscroll(1)
RGBlabel(2)
RGBscroll(2)
Label(0)
RGBlabel(3)
RGBscroll(3)
Label(1)
Label(2)
API Dithered
DitherBtn
ClipPalette
ColorForm
Xor_Mask
Xor Mask
Close_Button
Close

图 9.1　Palette 程序

产生喷涂填色的两端颜色，是通过选择 PicRGB 图片框中的颜色来实现的。然后通过 Palette 程序中的 RGBColor& 函数来找到两端颜色不同的红、绿、蓝分量，以实现在 PicBlend 图片框中的调色，产生喷涂填色。

在本程序中使用的喷涂填色的第二种方法是通过 API 库中的函数分离出一种颜色的红、绿、蓝分量。在绘出每个调色时采用的是 RGB 方法——即通过三基色来混合调色的方法。DitherBtn 按钮控制对象，通过 API 的 Draw_Blends 函数可以将喷涂填充的颜色直接绘于窗体。

在 Palette 程序中使用 API Color& 函数中的一个公式来获得一种颜色分离的红、绿、蓝

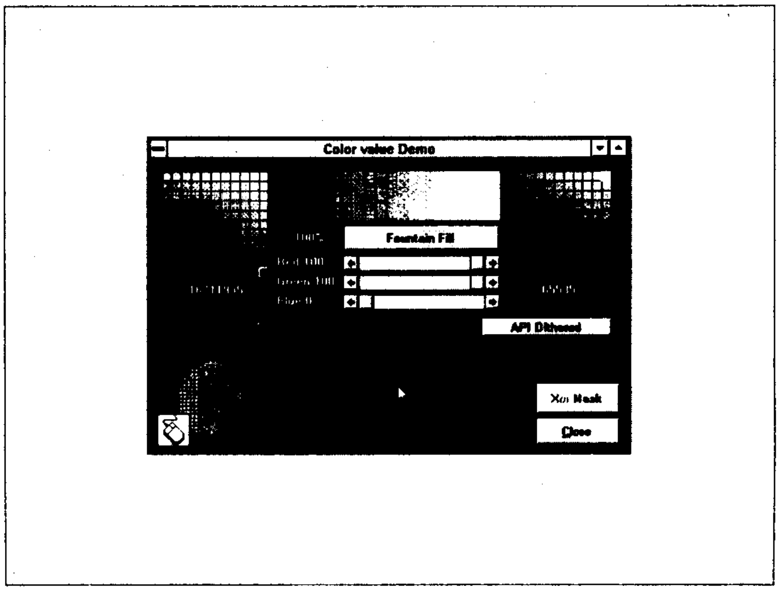


图 9.2　Palette 程序的屏幕显示

分量。最后，CreateCustomPalette 过程产生了两种颜色之间的各种色调（两种颜色是由程序员在 PicRGB 图片框中选择的）。

9.1　控制颜色的结构体

在 PALETTE.BAS 文件中定义了 TYPE 结构 LOGPALETTE 和 PALETTEENTRY。这两个结构用来定位与分离每种颜色的红、绿、蓝分量；如同 Visual Basic 中的 RGB 语句的功能一样。

使用 LOGPALETTE 和 PALETTENTRY 结构来定义颜色分量，可以应用 API 的调色板函数。在 Fountain Fill 程序中，使用了 API 的 Create Palette 函数来将一个调色板拷贝到剪贴板上，并把此调色板作为一个 Picture 资源调入了主窗体。如果你有 VGA 显示卡并能够提供 256 色或更多，则可以得到非常动人的调色色调。单击 Fountain Fill 按钮，就可以看到在抖色模式下同样的颜色效果（抖色模式：通过 16 种基本色构成的众多点纹来近似纯色）。

注意： 本程序假定使用的显示器为 256 色。假如使用的显示器超过 256 色，抖色模式的效果就看不到了。例如，一些 65536 的显示器在使用 Visual Basic 时只显示纯色。假如使用的 VGA 卡只支持 16 色，则将不能在剪贴板上建立一个调色板，这样，在本程序中提供了用鼠标来选择调色。

通过标题为 Xor Mask 的 Xor_Mask 按钮控制对象来实现对任一形状及大小物体的喷涂填充。

通过滚动条来显示单色。红、绿、蓝的每一颜色值的范围为 0～25，当 Check1 标题控制对象显示为 255%时，每一滚动条的标题会显示给定颜色的红、绿、蓝分量值；当 Check1 标题控制对象显示为 100%时，每一个颜色分量即被 2.55 除，所以显示的数值范围从 0～25 变成 0～100%。

表 9.1 给出了 Palette 程序的窗体及其控制对象的属性。

表 9.1　Palette 的窗体及其控制对象的属性

Object: Form
Object Name: Colorform
BackColor =&H00FF0000&　　Caption ="Color value Demo"
Height =4920　　Left =1245
ScaleMode =3' Pixel　　Top =1230
Width =7530

Object: Picture Box
Object Name: ClipPalette
AutoSize =－1 'True　　Enabled =0 'False
Height =510　　Left =120
Picture =Mouse icon　　Top =3960
Width =510

Object: Command Button
Object Name: Close_Button
Caption ="&Close"　　Height =435
Left =6000　　Top =4020
Width =1335

Object: Command Button
Object Name: Xor_Mask
Caption ="Xor Mask"　　Enabled =0 'False
Height =495　　Left =6000
Top =3480　　Width =1335

Object: Frame
Object Name: Frame1
BackColor =&H00000000&　　Height =2775
Left =120　　Top =120
Width =7155

Object: Command Button
Object Name: DitherBtn
Caption ="API Dithered"　　Enabled =0 'False
Height =315　　Left =5040
Top =2340　　Width =2055

Object: HScrollBar
Object Name: RGBscroll 3 indexed control
Height =255　　Index =0 To 3
LargeChange =35　　Max =255
SmallChange =4　　Width =2475

Object: Picture Box
Object Name: PicBlend
BackColor =&H000000FF&　　Height =795
Left =1800　　MousePointer =2 'Cross
Top =120　　Width =3555

Object: Picture Box
Object Name: PicRGB
Height =1680　　Index =0
Left =60　　ScaleHeight =110
ScaleMode =3 'Pixel　　ScaleWidth =110

```
Top          =60                   Width        =1680
```

Object:Picture Box
Object Name: PicRGB

```
Height       =1680                 Index        =1
Left         =5400                 ScaleHeight  =110
ScaleMode    =3 'Pixel             ScaleWidth   =110
Top          =60                   Width        =1680
```

Object:Command Button
Object Name: Fill_Button

```
Caption      ="Fountain Fill"      Enabled      =0 'False
Height       =435                  Left         =2880
Top          =900                  Width        =2475
```

Object:Check Box
Object Name: Check1

```
BackColor    =&H00000000&          Caption      ="100%"
ForeColor    =&H00FFFFFF&          Height       =375
Left         =1860                 Top          =900
Width        =915
```

Object:Label
Object Name: Label1

```
Alignment    =2 'Center            BackColor    =&H00FFFFFF&
BorderStyle  =1 'Fixed Single      ForeColor    =&H00FFFFFF&
Height       =315                  Index        =2
Left         =60                   Top          =2340
Width        =4875
```

Object:Label
Object Name: RGBlabel

```
BackColor    =&H00000000&          Caption      ="Red 0"
ForeColor    =&H00FFFFFF&          Height       =255
Index        =1                    Left         =1860
Top          =1380                 Width        =1035
```

Object:Label
Object Name: RGBlabel

```
BackColor    =&H00000000&          Caption      ="Green 0"
ForeColor    =&H00FFFFFF&          Height       =255
Index        =2                    Left         =1860
Top          =1680                 Width        =1035
```

Object:Label
Object Name: RGBlabel

```
BackColor    =&H00000000&          Caption      ="Blue 0"
ForeColor    =&H00FFFFFF&          Height       =195
Index        =3                    Left         =1860
Top          =1980                 Width        =1035
```

Object:Label
Object Name: Label1

```
Alignment    =2  'Center           BackColor    =&H00FFFFFF&
BorderStyle  =1  'Fixed Single     ForeColor    =&H00FFFFFF&
Height       =435                  Index        =0
Left         =60                   Top          =1800
Width        =1695
```

Object:Label

```
Object Name: Label1
    Alignment      =2' Center            BackColor     =&H00FFFFFF&
    BorderStyle    =1' Fixed Single      ForeColor     =&H00FFFFFF&
    Height         =435                  Index         =1
    Left           =5400                 Top           =1800
    Width          =1695
```

9.2　Palette 程序的声明

清单9.1给出了在 Palette 程序中使用的默认变量。

清单9.1　本程序中使用的默认变量

```
Declarations
Dim ColorIndex&(0 To 120) 'stores each color value
Dim Pic1Red, Pic1Green, Pic1Blue As Integer 'RGB of PicRGB(0)
Dim Pic2Red, Pic2Green, Pic2Blue As Integer 'RGB of PicRGB(1)
```

Dim ColorIndex&(0 To 120)：有两个 PicRGB 图片框控制对象，每一个有121色。ColorIndex& 数组用来存储当前被选中的 PicRGB 图片框中的每一种颜色，每当用户选择一颜色以后，PicRGB_Paint 事件更新本数组一次（本数组通过十六进制数存储每一种颜色）。

Dim Pic1Red, Pic1Green, Pic1Blue：位于屏幕左侧的 PicRGB 图片框控制对象中的颜色如被选中，这三个变量即被用来存储该颜色的红、绿、蓝三分量。当用户从 PicRGB 中选择一颜色，在 MouseDown 事件中，通过一公式计算得到该颜色的红绿蓝分量并存入 Pic1Red，Pic1Green 及 Pic1Blue 中，以用于接下来的处理中。

Dim Pic2Red, Pic2Green, Pic2Blue：同 Pic1Red 等变量，只不过这三个变量用于右侧的 picRGB 图片控制对象。

9.3　Form_Resize 过程

Palette 程序中的 Form_Resize 过程用来置标题控制对象的背景色为黑色。这个过程如清单9.2所示。

清单9.2　置标题控制对象的背景为黑色

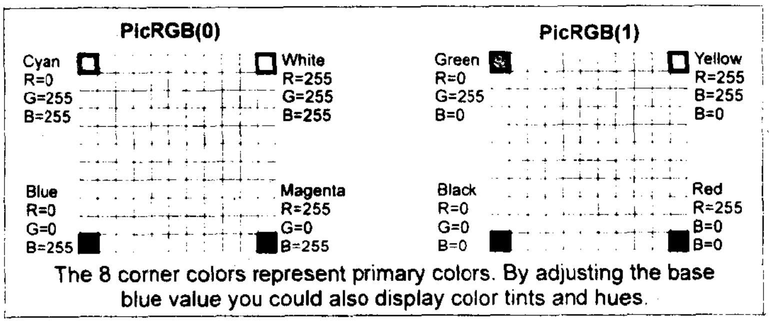
```
Sub Form_Resize ()
    Label1(0).BackColor = Frame1.BackColor
    Label1(1).BackColor = Frame1.BackColor
    Label1(2).BackColor = Frame1.BackColor
End Sub
```

Label1(index).BackColor=Frame1.BackColor：将每一个标题控制对象的背景置为黑色以便与窗体的背景色相同，这样可以使你更清楚地看见标题。

9.4 两个主调色板

两个 PicRGB 图片框调色板通过 PicRGB_Paint 事件来显示出 121 种颜色。第一个调色板(PicRGB(0))中每一种颜色的蓝分量为常数 255；第二个调色板(PicRGB(1))中每一种颜色的蓝分量为常数 0，在 paint 事件中首先处理 PicRGB(0)接着处理 PicRGB(1)。通过 If... Else 语句可以改变两个图片框的蓝色分量值。

下一步是在两个调色板上平均地分配颜色范围，在 PicRGB(0)调色板中，基本色为青、白、蓝和洋红。在四个基本色之间，通过将红、绿分量值的递增或递减产生出中间的渐变的颜色。在 PicRGB(1)调色板中，基本色为绿、黄、黑和红，同样通过红、绿分量值的递增或递减产生四种基本色中间的颜色(见图 9.3)。


PicRGB(0)
PicRGB(1)
Cyan
R=0
G=255
B=255
White
R=255
G=255
B=255
Blue
R=0
G=0
B=255
Magenta
R=255
G=0
B=255
Green
R=0
G=255
B=0
Yellow
R=255
G=255
B=0
Black
R=0
G=0
B=0
Red
R=255
B=0
B=0

The 8 corner colors represent primary colors. By adjusting the base blue value you could also display color tints and hues.

图 9.3 基本颜色的产生

8 个角上的颜色即为基本色。通过调整蓝分量的值，可以显示出各种颜色的色度。如果基本色之间按照八等分变化，你可以得到绝大多数颜色，而且，如果可以增加或减少蓝色的值(从 255 至 0)，你就可以产生不同色度的黄、白、蓝、黑来显示 16 777 216 种颜色组合。

设定 PicRGB 图片框为 11×11，在每个调色板中，得到了均分的 121 种颜色(如清单 9.3 所示)。

清单 9.3 在每 PicRGB 图片框控制对象中画出 121 种颜色

```
Sub PicRGB_Paint (Index As Integer)
    PicRGB(Index).Scale (0, 0)-(11, 11)
    Colordown = 100: Colorup = 0 'color start values
    If Index = 0 Then BaseColor = 100 * 2.55 Else BaseColor = 0 'base color blue
For Y1 = 0 To 10
    For X1 = 0 To 10
        Red1 = Colorup * 2.55: Green2 = colordown * 2.55: Blue3 = BaseColor
        ''Label1(2).Caption = "Red " + Format $ (Red1) + " Green " +
            Format $ (Green2) + " Blue " + Format $ (Blue3)
        PicRGB(Index).Line(X1,Y1)-(10+X1,10+Y1),RGB(Red1,Green2,Blue3),BF
        ColorIndex(i) = RGB(Red1, Green2, Blue3): i = i + 1
        PicRGB(Index).Line (X1, Y1)-(10 + X1, 10 + Y1), , B
        Colorup = Colorup + 10
    Next X1 'next column to the right
        Colorup = 0: colordown = colordown - 10
Next Y1 'next row below
```

```
End Sub
```

PicRGB(Index).Scale(0,0)－(11,11)：设定每个图片框为 11＊11 的网格。

Colordown＝100;Colorup＝0：给出了增加或减少红绿分量的起始点。

If Index＝0 Then BaseColor＝100＊2.55 Else BaseColor＝0：两个 PicRGB 图片框中，当 Index＝0 即第一个调色板时，基本的蓝分量设定为 255，在 Index＝1 时，蓝色分量被设定为 0。

For Y1＝0 To 10：在调色板设定新的一行的颜色。

For X1＝0 To 10：设定每一行中的 11 种颜色。

Red1＝ColorUp＊2.55:Green2＝Colordown＊2.55:Blue3＝Bascolor：在每一小格中，红绿分量都相对上一个值增加或减少以得到正确的颜色。

“Label1(2).Caption＝"Red"＋Format $ (Red1)＋etc.：为了看到对每一小格颜色的绘制，移去注释符号“' ”，并在此语句前加上 Stop，当暂停时，按 F5 继续执行，这样可以看见每一小格的颜色绘制。

PicRGB(Index).Line(X1,Y1)－(10＋X1,10＋Y1),etc.：本语句绘出每一个小格并在其中填以颜色，改变 Line 中的参数，可以改变小格的大小。

ColorIndex(i)＝RGB(Red1,Green2,Blue3)：将当前的调色板中 121 色存入 ColorIndex 中，通过十六进制数存储每一种颜色。当用户使用鼠标选择一种颜色时，PicRGB_Paint 事件被触发一次，通过 ColorIndex 将调色板重绘，i＝i＋1 循环得到从 1 到 121 的标号。

PicRGB(Index).Line(X1,Y1)－(10＋X1,10＋Y1),,B：为每一小格绘制黑色边界框。

Colorup＝Colorup＋10：在循环起始时，Colorup 为 0，并通过此语句实现逐步增至 100，以实现红、绿色分量的递增。

Next X1：循环继续(在一行内)。

Colorup＝0:Colordown＝Colordown－10：在每一行结束后，Colorup 变量重置为 0，并且 Colordown 减去 10。

Next Y1：开始下一行，且循环继续。

9.5 从调色板中选择一颜色

PicRGB_MouseDown 事件在选中的颜色框上绘出白色的高亮框，并从 ColorIndex 数组中得到该颜色的红、绿、蓝三色分量。该三分量的值即被显示于 RGBScroll 标题控制对象中。实现此功能的程序如清单 9.4 所示。

清单 9.4　将选中的颜色高亮显示并得到颜色值

```
Sub PicRGB_MouseDown (Index As Integer, Button As Integer, Shift As Integer, X As Single,
Y As Single)
Xor_Mask.Enabled = False
    PicRGB_Paint Index
    X = Fix(X): Y = Fix(Y)
PicRGB(Index).DrawWidth = 2
PicRGB(Index).Line (X, Y)-(X + 1, Y + 1), QBColor(15), B
PicRGB(Index).DrawWidth = 1
```

```
        Temp = X + (Y * 11)
        Label1(Index).Caption = ColorIndex(Temp)
    'R1 = (ColorIndex(Temp) And &HFF)
    'G2 = (ColorIndex(Temp) And &HFF00&) \ 256
    'B3 = (ColorIndex(Temp) And &HFF0000) \ 65536
    'R1 = ColorIndex(Temp) Mod 256
    'G2 = ColorIndex(Temp) \ 256 Mod 256
    'B3 = ColorIndex(Temp) \ 65536 Mod 256
    If Index = 0 Then Pic1Red = R1: Pic1Green = G2: Pic1Blue = B3
    If Index = 1 Then Pic2Red = R1: Pic2Green = G2: Pic2Blue = B3
            RGBScroll(1).Value = Val(R1)
            RGBScroll(2).Value = Val(G2)
            RGBScroll(3).Value = Val(B3)
    End Sub
```

PicRGB_Paint Index：重绘 PicRGB 中的每一颜色小格，这一方法要比用 Xor 来清除上一个白色的框容易得多。

X＝Fix(X)：Y＝Fix(Y)：PicRGB 调色板被设定为(0,0)至(11,11)。同时，X 和 Y 被定为 Single(整数)，Fix 语句将 X 和 Y 的整数值传递过去。

PicRGB(Index).DrawWidth＝2：改变宽度以绘出一个厚的白色框。

PicRGB(Index).Line(X,Y)－(X＋1,Y＋1),QBColor(15),B：在选中的颜色格上绘出白色框。

PicRGB(Index).DrawWidth＝1：重置宽度为 1。

Temp＝X＋(Y＊11)：这个简单的公式给出选中的小格是第几号格。

Label1(Index).Caption＝ColorIndex(Temp)：ColorIndex 数组中的每一个索引号对应一个颜色小格。Label1(Index).Caption 将显示出该颜色的值。

接下来的公式是用来从十六进制颜色值中得到 R、G、B 三个基本色值，一个十六进制颜色值是指一个长整型通过十六进制数表示一种颜色，Hexa 意为 6，decimal 意为 10，故 Hexadecimal 意为十六进制数，在十六进制中，使用 0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F 表示 16 个数字且 A 相当于十进制的 10，E 相当于 14，F 相当于 15，以此类推。在一个数字前加上 &H 符号即表示该数为十六进制。

十六进制的基本理论如下：任意一个十六进制数的值等于其各位的数值乘以位置值。

十六进制数的位置值从右至左分别为 1，16(例如，从右至左依次为 16^0，16^1，16^2 和 16^3，等等)，如值 FF 等于十进制 255，计算为：15×16＋15＝255；而值 FF00 计算为：(15×4096)＋(15×256)＋(0×16)＋(0×1)＝65280；值 FF0000 计算为：(15×1,048,576)＋(15×65,536)＋(0×4096)＋(0×256)＋(0×16)＋(0×1)＝16,711,680。

可见，很短的十六进制数即可表示相当大的十进制数，红、绿、蓝三色的十进制范围为 0～255，用十六进制数可表示为 0～FF；十六进制数 &HFF 为红色(255)，&HFF00 为绿色(65280)，&HFF0000 为蓝色(16,711,680)，每一个 RGB 值用六位数表示，即十六进制数：&Hrrggbb，存储于 ColorIndex 中。

（续）

为了从 ColorIndex 中分离出各个颜色值，需使用 AND 运算符，可以参见“Visual Basic language reference guide，Operators”颜色值的十进制值范围是 0～16，7777，216，使用 AND 符，从 ColorIndex 的各位中提取出红、绿、蓝三分量的值，语句如下所示：

```
R1=(ColorIndex(Temp) AND &HFF)
G2=(ColorIndex(Temp) AND &HFF00\256)
B3=(ColorIndex(Temp) AND &HFF0000\65536)
```

如果使用 Mod 运算符，可以表示成如下语句：

```
'R1=(ColorIndex(Temp) Mod 256
'G2=(ColorIndex(Temp) AND \256 Mod 256
'B3=(ColorIndex(Temp) \65536 Mod 256)
```

If Index=0 Then Pic1Red=R1：这些变量存储了左侧 PicRGB(0)图片框中颜色的 R、G、B 分量，Pic1Red，Pic1Green 和 Pic1Blue 被用来作为喷涂填充的第一个颜色。

If Index=1 Then Pic2Red=R1：这些变量存储了右侧 PicRGB(1)图片框中颜色的 R、G、B 分量，Pic2Red，Pic2Green 和 Pic2Blue 被用来作为喷涂填充的最后一个颜色。

RGBScroll(i). Value=Val(R1)，Val(G2)，Val(B3)：通过改变 RGBScroll 的值，被选中的颜色分量将显示于 PicBlend 图片框中，并且各 R、G、B 分量值也显示于滚动条上，每一滚动条的最小值为 0，最大值为 255。

9.6　使 Fountain Fill 控制对象生效

只有在每一个 PicRGB 调色板上均有一颜色被选中以后，Fill_Button，DitherBtn 以及 Picture1 控制对象才被激活，清单 9.5 给出了 PicRGB_MouseUp 例程，这个程序控制着许多按钮的开启。

清单 9.5　控制按钮的开与关

```
Sub PicRGB_MouseUp (Index As Integer, Button As Integer, Shift As Integer,
    X As Single, Y As Single)
    If Label1(0).Caption <> "" And Label1(1).Caption <> "" Then
        Fill_Button.Enabled = True: DitherBtn.Enabled = True
        ClipPalette.Enabled = True
    Else
        Fill_Button.Enabled = False: DitherBtn.Enabled = False
        ClipPalette.Enabled = False
    End If
End Sub
```

If Label1(0). Caption<>""：当两个调色板中均有一颜色被选中，则每个调色板下的标题控制对象值为十六进制颜色值，此时，各按钮均可被使用；当有一个标题控制对象为空时，各按钮将被置为不可使用。

9.7 API 调色板

当某一事件需要使用 Draw_Blends 例程时，APIColor& 函数即被用来画一个 API 调色板（API Color Palette）。清单 9.6 给出了 APIColor& 函数的程序。当一个 API Palette 建立以后，每一个颜色均存放在数组 CustomPalette 中，这个调色板是通过 Type 结构的 LOGPALETTE（PALETTE.BAS）定义的，在这个结构中，使用子程序 palPalEntry 来定义每种颜色的红、绿、蓝三分量，palPalEntry 使用 PALETTENTRY 结构来定义一种颜色的红、绿、蓝分量。

清单 9.6　把一种颜色的 R、G、B 分量合并

```
Function APIcolor& (cell)
    Dim Color As PALETTEENTRY
    Color = customPalette. palPalEntry(cell)
    RGBcolors& = Asc(Color. peRed)
    RGBcolors& = RGBcolors& Or (Asc(Color. peGreen) * 256&)
    RGBcolors& = RGBcolors& Or (Asc(Color. peBlue) * 256& * 256&)
    APIcolor& = RGBcolors&
End Function
```

你可以用语句 CustomPalette. palPalEntry(cell)来检查每一个单独的颜色。其中，变量 cell 用来表示一种颜色的索引号。

每一种颜色值都通过 ANSI 的字符串形式存放，所以，必须使用 Visual Basic 的 ASC 语句来将 ANSI 字符串转化为数值，每一个数值再接着作为 R、G、B 分量被合并起来，形成一种颜色，这个过程与清单 9.4 给出的分离每一个红、绿、蓝分量的过程很类似：但不像清单 9.4 那样使用 AND 运算符及除法，而是采用 OR 运算符及乘法来把各分量合并起来。

最后的结果将是一个十进制整数并被存放于变量 APIColor& 中，这个值将被应用于语句：Line(StrartCell,0)-(EndCell,ScaleHeight),APIColor&(r),BF 中，用来填充每一个小格(cell)。（注：此语句出现在 Draw_Blends 子程序中，见清单 9.12）。

9.8 显示颜色值的范围

Check1 检查框控制对象触发 RGBscroll_Scroll 事件，通过 RGBlabel 标题显示出每个颜色分量的值，该值的显示标度为 100%或 255%。

清单 9.7　选择滚动条的尺度为 100 或 255

```
Sub Check1_Click ()
    For r = 1 To 3
        RGBscroll_Change (r)
    Next r
End Sub
```

RGBscroll_Change(r)：通过三个水平的滚动条控制对象来表示一种颜色。每一条滚动条表示 R、G、B 三分量的一个，每一分量的值为 0 至 255，当红为 0，绿为 0，蓝为 0 时，显

示的颜色为黑；当红为255，绿为255，蓝为255时，显示的颜色为白。改变三个分量的值可以对基色进行色调的修改。

使用Check1_Click过程来显示RGB三色(100%或255%两种尺度，其计算在RGBscroll_Change事件中，清单见9.15。

9.9　显示一个纯色的调色板

你可以使用Visual Basic的剪贴板属性把一个自定义的API Palette拷贝入一个窗体或图片框控制对象中，如果VGA显示卡为256色或更强，就可以产生纯色的喷涂填充而不是采用抖色，清单9.8给出了ClipPalette_MouseDown例程，此段程序将一个自定义调色板拷贝到剪贴板上。

清单9.8　将一个自定义调色板拷贝到剪贴板上

```
Sub ClipPalette_MouseDown (Button As Integer, Shift As Integer, X As Single,
    Y As Single)
    ColorForm.DrawMode = 13
        CreateCustomPalette
    API = OpenClipboard(ColorForm.hWnd)
If API = 0 Then MsgBox "clipboard error": End
    API = SetClipboardData(CF_PALETTE, CreatePalette1(customPalette))
    API = CloseClipboard()
    ColorForm.Picture = Clipboard.GetData(CF_PALETTE)
    Xor_Mask.Enabled = True
End Sub
```

CreateCustom Palette：你必须首先定义一个新的用户调色板来显示你希望实现的调色范围。CreateCustomPalette过程产生两种颜色之间20等分的各种渐变色。两种起始，终止颜色是从PicRGB图片框控制对象中选择的。

API=OpenClipboard(ControlForm.hWnd)：在CustomPalette数组中定义一个调色板后，打开Windows的剪贴板，以便把CustomPalette数组拷贝到主窗体，这有点像调入一个256色的Visual Basic的bitmap文件一样。

If API=0 Then MsgBox"clipboard error":End：假如剪贴板打开失败，API的OpenClipboard函数将返回0值于调用API的变量中，过程结束。

API=SetClipboardData(CF_PALETTE,CreatePalette1(CustomPalette))：如果剪贴板成功地打开，用API函数CreatePalette将定义于CustomPalette数组的20种颜色进行拷贝，CF_PALETTE标志使剪贴板将接下的API CreatePalette信息按照调色板数据的形式格式化。

ColorForm.Picture=ClipBoard.GetData(CF_PALETTE)：调色板(Palette)现在可以作为Visual Basic的标准图形资源来调用，你应该使用标志CF_PALETTE来指明调入的数据为调色板(因为剪贴板中可以装有文本块)，以便和其他应用程序的位图、图元文件等信息加以区分)。

Xor_Mask.Enabled=True：Xor Mask按钮被置为可用，这样，就可以对一个较复杂形

状的物体进行纯色的喷涂填充。

当一个调色板从剪贴板上拷入一个窗体或一个图片框中后，在该窗体或图片框中就可以使用调色板上的颜色，Draw_Blends 过程通过 Visual Basic 语句实现主窗体 ColorForm 的喷涂填充与调色，见清单 9.9。

清单 9.9　绘制由剪贴板提供的纯色 Palette

```
Sub Clip_Button_MouseUp(Button As Integer,Shift As Integer,X As Single,Y As Single)
    Draw_Blends
End Sub
```

9.10　创建一个自定义调色板(Custom Palette)

清单 9.10 给出了 CreateCustomPalette 函数的程序清单。该函数用来产生两种颜色（从 PicRGB 中选择的）之间的渐变调色，该函数存储颜色值的方式与 RGBColor 函数中采用的方式相同，在 PALETTE.BAS 文件中定义的 Type 结构 LOGPALETTE 和 PALETTEENTRY 用来定位和分解每种颜色的红、绿、蓝分量，其方法与 Visual Basic 的 RGB 语句相同。

清单 9.10　利用喷涂填充调色产生一个自定义调色板

```
Sub CreateCustomPalette ()
Temp1 = Pic1Red: Temp2 = Pic1Green: Temp3 = Pic1Blue
customPalette.palVersion = &H300 'Window version
customPalette.palNumEntries = APIcells 'total number of colors
For cellnumber = 0 To APIcells - 1
    Distance = Pic1Red - Pic2Red
    Red1 = Temp1 - ((Distance) \ (APIcells)): Temp1 = Red1

    Distance = Pic1Green - Pic2Green
    Green1 = Temp2 - ((Distance) \ (APIcells)): Temp2 = Green1

    Distance = Pic1Blue - Pic2Blue
    Blue1 = Temp3 - ((Distance) \ (APIcells)): Temp3 = Blue1

    customPalette.palPalEntry(cellnumber).peRed = Chr$ (Red1)
    customPalette.palPalEntry(cellnumber).peGreen = Chr$ (Green1)
    customPalette.palPalEntry(cellnumber).peBlue = Chr$ (Blue1)
    customPalette.palPalEntry(cellnumber).peFlags = Chr$ (0)
Next cellnumber
End Sub
```

Temp1＝Pic1Red:Temp2＝Pic1Green:Temp3＝Pic1Blue：将从左侧 PicRGB 图片框控制对象中选择的初始 R、G、B 分量存储下来。

CustomPalette.palVersion＝&H300：为了兼容性的原因，LOGPALETTE 结构需要 Windows 的版本号。

CustomPalette.palNumEntries＝APIcells：LOGPALETTE 结构需要自定义调色板中颜色小格的总数。

For cellnumber＝0 To APIcells－1：从索引号为 0 的颜色小格至 APIcell－1 的颜色小

格，循环执行以下代码语句。

Distance＝Pic1Red－Pic2Red：从左侧 PicRGB(0)中选择的颜色值与从右侧 PicRGB(1)中选择的颜色值之间的差。

Red1＝Temp1－((Distance)\(APIcells))：从当前色渐变至另一色，Red1 变量减少一定值。通过本语句实现颜色渐变。

Temp1＝Red1：置 Temp1 为当前色，下一个颜色小格的颜色以此为基准，通过公式 Red1＝Temp1－((Distance)\(APIcells))求得。

Next cellnumber：进行下一循环，直至自定义调色板画完为止。

9.11　PALETTE 和 PALETTEENTRY

新的调色板被称为 CustomPalette，并以 Type LOGPALETTE 结构存放在文件 PALETTE.BAS 中，LOGPALETTE 结构的第三和最后细目中存放着每种颜色的红、绿、蓝三分量的值，每种颜色的 RGB 分量通过 PALETTEENTRY 结构来区分，在 Type PALETTEENTRY 结构中，每一个值都作为一个 ANSI 字符来存放，这样，可以使用 Visual Basic 的 Chr $ 语句将一个十进制数转化为 ANSI 字符，下面的一行程序代码给出了如何将 Red1 的颜色值转化成 ANSI 字符的形式并存储：

```
CustomPalette. palPalEntry(cellnumber). peRed=Chr $ (Red1)
```

9.12　在调色板中使用抖色

单击 API Dither 按钮，窗体内将显示抖色的调色板，抖色是指通过组成基本 16 色的各种点纹来模拟的纯色，清单 9.11 给出了 DitherBtn_Click 事件的程序码：

清单 9.11　显示抖色的调色板

```
Sub DitherBtn_Click ()
    ColorForm. Picture = LoadPicture()
    Xor_Mask. Enabled = True
        ColorForm. BackColor = QBColor(0)
    DrawMode = 7 'Xor pixel on a black background
    Draw_Blends    ' will restore original color
    DrawMode = 13 'restore
End Sub
```

ColorForm. Picture＝LoadPicture()：主 ColorForm 窗体中可能包含有纯色剪贴板调色板，我们可以通过调入一个空白图片来清除窗体内原来的纯色剪贴板调色板。

Xor_Mask. Enabled＝True：Xor_Mask 按钮被置为可以使用；此时，可以单击 Xor Mask 按钮以实现对复杂物体的抖色的喷涂填充。

ColorForm. BackColor＝QBColor(0)：用 Visual Basic 的 Line 语句在窗体中画出与窗体等高且紧挨着排列的小框，每一个小框用喷涂色填充。不过 Line 语句绘制的小框有一个象素宽的边框线，应删去它们，否则边框线会在填充色之间显示出来。

> 去掉这些不需要的线有一个小窍门，即用 Xor DrawMode(7)和 Transparent DrawStyle(5)两语句，此时，如果背景色为黑，就可以消除不需要的边界线条，更详尽的信息请见《Visual Basic 程序员指南》一书。

DrawMode=7：将绘图模式改变为 Xor_Pen。

Draw_Blends：调用 Draw_Blends 子程序，用 Visual Basic 的 line 语句在主 ColorForm 控制对象中画出喷涂填充。

DrawMode=13：将绘图模式变回正常(copy_Ren)。

9.13 绘制调色板上的每一颜色

当数组中已经存放好表示每一种调色的值时，就应该在窗体中将这些颜色绘制出来。请参见清单 9.12 给出的程序。

清单 9.12 通过 API 的结构体画出喷涂填充调色的效果

```
Sub Draw_Blends ()
CreateCustomPalette
Scale (0, 0)-(APIcells, 1)
For r = 0 To APIcells - 1
            'For RR = 1 To 15000: Next RR
StartCell = (ScaleWidth * r) \ APIcells
EndCell = (ScaleWidth * (r + 1)) \ APIcells
    ColorForm. FillStyle = 0
    ColorForm. DrawStyle = 5 'Transparent
    ColorForm. FillColor = APIcolor&(r)
    Line (Startcell, 0)-(Endcell, ScaleHeight), , B
    'PX = Screen. TwipsPerPixelX
    'StartCell = ((Width \ PX) * r) \ APIcells
    'EndCell = ((Width \ PX) * (r + 1)) \ APIcells
    'API = Rectangle(ColorForm.hDC, StartCell - 1, 0, EndCell, Height)
Next r
ColorForm. DrawStyle = 0 'Reset to Solid
End Sub
```

CreateCustomPalette：本函数产生喷涂填充中的每一种颜色，这些颜色介于在 picRGB 图片框中选择的两种颜色之间，颜色值通过本子程序(见清单 9.10)用与 RGBcolor& 函数(见清单 9.14)类似的 Type 结构存储于数组中。

Scale(0,0)-(APIcells,1)：将主 ColorForm 窗体控制对象缩放成一个大调色板。

For r=0 To APIcells-1：开始循环，在本循环内将绘制喷涂填充中每个颜色小格。

'For RR=1 To 20000:Next RR：如果要看每一个小格是怎样绘出的，可以用本语句。

Startcell=(ScaleWidth * r)\APIcells：因为窗体已进行过缩放，用本公式计算每一个颜色小格的绘制起始点。

EndCell=(ScaleWidth * (r+1)\APICells：因为窗体已进行过缩放，用本公式计算每一个颜色小格的绘制终止点。

ColorForm. FillStyle=0：确定填充方式为 solid。

ColorForm. DrawStyle=5：置绘制模式为透明。请参考 DitherBtn_Click 事件（清单 9.11）弄清置绘制模式为透明（Transparent）的原因。

ColorForm. FillColor=APIcolor&(r)：调用 APIcolor& 子程序（见清单 9.6）来为后面的 Line 语句分配正确的填充色。

Line(Startcell,0)-(Endcell,ScaleHeight),APIcolor&(r)),BF：采用 Visual Basic 的 Line 语句来绘制调色板上每一颜色小格的形状（方块形），填充颜色由本语句中的 APIcolor&（见清单 9.6）函数确定。

'API=Rectangle：去掉' 号后，这个语句将采用 API 的 Rectangle 函数来实现喷涂填充。但用 API 函数进行喷涂填充速度太慢，如果想使用本语句，注意将上一句（Line…）用' 注释掉。

Next R：继续循环直至所有颜色均被绘制出来。

9.14　Fountain Fill 按钮

清单 9.13 给出 Fill_Button_Click 事件的程序，该子程序与 Draw_Blends（见清单 9.12）过程很类似；这两个子程序的不同点在于 Line 语句中颜色选项的安排不同。

清单 9.13　通过 Visual Basic 的 RGB 语句画出喷涂填色

```
Sub Fill_Button_Click ()
If Label1(0).Caption = "" Or Label1(1).Caption = "" Then Exit Sub
        PicBlend. Scale (0, 0)-(numcolors, 1)

    For r = 0 To numcolors - 1
        'For RR = 1 To 30000: Next RR
        Startcell = (PicBlend. ScaleWidth * r) / numcolors
        Endcell = (PicBlend. ScaleWidth * (r + 1)) / numcolors
        PicBlend. Line (Startcell, 0)-(EndCell, PicBlend. ScaleHeight), RGBcolor&(), BF
    Next r
End Sub
```

PicBlend. Scale(0,0)-(numcolors,1)：PicBlend 图片框规定了喷涂填充色中填多少种颜色（numcolors）。在 PALETTE. BAS 文件中，numcolor 的值被定义为常数 20。

For r=0 To numcolor-1：从 0 递增到 19，在 PicBlend 中画出全部的 20 种颜色。

Startcell=(PicBlend. ScaleWidth * r)/numcolors：每一颜色框的左侧位置值。

Endcell=(PicBlend. ScaleWidth * (r+1)/numcolors：每一颜色框的右侧位置值。

PicBlend. Line(Startcell,0)-(Endcell,PicBlend. ScaleHeight)：在 PicBlend 图片框中绘出每一个颜色框，并在其中填充上颜色，填充的颜色由 RGBcolor& 函数控制（见清单 9.14）。函数 RGBcolor& 用 RGB 值产生喷涂填充色。

Next r：重复上述过程，填充 PicBlend 图片框中的下一颜色框。

9.15　使用 RGB 颜色值来建立一个调色板

为喷涂填充提供颜色控制有二种方法，RGBColor& 函数是其中之一。第二种方法是由

.CreateCustomPalette 子程序提供的（见清单 9.10）。清单 9.14 给出 RGBColor& 函数的程序，该子程序计算出 PicRGB(0)与 PicRGB(1)之间的 18 种颜色，这样就提供了喷涂填充的 20 色。

清单 9.14 使用 Visual Basic 的 RGB 语句来调色

```
Function RGBcolor& ()
Static counter, Temp1, Temp2, Temp3
counter = counter + 1
Select Case counter
    Case 1
        ColorCell& = Val(Label1(0).Caption)
        Temp1 = Pic1Red: Temp2 = Pic1Green: Temp3 = Pic1Blue
    Case numcolors
        ColorCell& = Val(Label1(1).Caption)
        counter = 0: Temp1 = 0: Temp2 = 0: Temp3 = 0
    Case Else
        Distance = Pic1Red - Pic2Red
        Red1 = Temp1 - ((Distance) \ (numcolors - 2)): Temp1 = Red1
        Distance = Pic1Green - Pic2Green
        Green1 = Temp2 - ((Distance) \ (numcolors - 2)): Temp2 = Green1
          Distance = Pic1Blue - Pic2Blue
        Blue1 = Temp3 - ((Distance) \ (numcolors - 2)): Temp3 = Blue1
        ColorCell& = RGB(Red1, Green1, Blue1) 'color of cell
End Select
RGBcolor& = ColorCell&
End Sub
```

counter=counter+1：计数器，每次为 Fountain fill 填充一色，则 Counter 加 1。

Select Case counter：根据 counter 值，选择 Case 语句。

Case 1：第一种颜色，即从 PicRGB(0)中选择的颜色。

ColorCell&=Val：PicRGB(0)图片框下面显示的 RGB 值，它是喷涂填充的第一个颜色。

Temp1=Pic1Red:Temp2=Pic1Green:Temp3=Pic1Blue：存储在 PicRGB(0)图片框控制对象中选择的红、绿、蓝三色分量。Pic1Red，Pic1Green 和 Pic1Blue 三色分量是在第一次选取该颜色时由 PicRGB_MouseDown 事件计算出来的。

Case numcolors：最后一种颜色，即从 PicRGB(1)中选择的颜色。

ColorCell&=Val：PicRGB(1)图片框下面显示的 RGB 值，它是喷涂填充的最后一个颜色。

counter=0:Temp1=0:etc.：喷涂填色完成，重新设置所有的默认值。

Case Else：介于 Case 1 与 Case numcolor 之间的各种颜色均由本段程序控制。

Distance=Pic1Red-Pic2Red：在左边 PicRGB(0)图片框和右边 PicRGB(1)图片框所选的两种颜色值之差。

Red1=Temp1-((Distance)\(APIcells))：从当前色渐变至另一色，Red1 变量减少或增加一定值，通过本语句实现颜色渐变。

Templ＝Red1：置 Templ 为当前色，下一个颜色小格以此为基准，通过公式 Red1＝Templ－((Distance)\(APIcells))求得。

ColorCell&＝RGB：为 ColorCell& 变量提供新的 RGB 变量。

RGBcolor&＝ColorCell&：函数名的返回值为 Visual Basic 的 Line 语句中 RGBcolor 变量的值。

9.16　通过滚动条来改变颜色

RGBscroll_Scroll 事件通过每个滚动条的值在 PicBlend 图片框中显示颜色，并且在 RGBlabel 标题控制对象中显示出每个颜色分量的数值。清单 9.16 给出 RGBscroll_Scroll 过程，该过程是由 RGBscroll_Change 调用的(见清单 9.15)。

清单 9.15　通过滚动条显示单独的颜色

```
Sub RGBscroll_Change (Index As Integer)
    RGBscroll_Scroll Index
End Sub
```

RGBscroll_Scroll Index：调用 scroll 事件来显示颜色。

清单 9.16　显示一种颜色的红、绿、蓝分量值

```
Sub RGBscroll_Scroll (Index As Integer)
    PicBlend.BackColor = RGB(RGBScroll(1).Value,
          RGBScroll(2).Value, RGBScroll(3).Value)
    If Check1.Value = 1 Then
        ScaleValue = RGBScroll(Index).Value: Check1.Caption = "255%"
    Else
        ScaleValue = Int(RGBScroll(Index).Value / 2.55)
        Check1.Caption = "100%"
    End If
RGBlabel(Index).Caption = Choose(Index,"Red","Green","Blue")+ScaleValue
End Sub
```

PicBlend.BackColor＝RGB：调用滚动条的值，直接在 PicBlend 图片框中显示颜色。滚动条的取值范围为 Min＝0，Max＝255，其中 255 为允许的最大颜色值。

If Check1.Value＝1 Then：若选取选择框控制对象，则 ScaleValue 变量的值与滚动条的当前值相等，此时 RGBLabel 显示的颜色分量值范围为 0～255%。

Else：若选择框控制对象未被选中，则 ScaleValue 变量的值等于当前滚动条的值除以 2.55，此时 RGBlabel 显示的颜色值范围为 0～100%。

RGBlabel(Index).Caption：在 RGBlabel 中显示各颜色分量的值。

9.17　对一个复杂物体进行喷涂填色

通过 Xor_Mask 按钮可以对任何形状的物体进行喷涂填色(见清单 9.17)。首先以 Xor 模式画上抖色，然后再在其上画上五个连环的圆(这五个圆以黑色实线画上)，当我们再次以

Xor 模式进行抖色处理时，黑的圆圈将吸收掉真正的喷涂色。效果如图 9.2 所示。

清单 9.17　对各种形状的物体进行喷涂填色

```
Sub Xor_Mask_Click ()
    BackColor = &HFF0000 'remake out if you wish
    DrawMode = 7 'Xor dithered colors
            Draw_Blends
    Scale
    DrawMode = 1 'draw any shape solid black
    FillStyle = 0
        For r = 0 To 5000 Step 1000
            Circle (1000 + r, Height - 1200), 600
        Next r
    DrawMode = 7 'Xor again it fill shape with blends
            Draw_Blends
    DrawMode = 13 'restore
End Sub
```

BackColor=&HFF0000：改变窗体的背景色以清除原有的背景图案，可以用'符号注释掉本语句，也可用 Cls 语句。

DrawMode=7：置绘制模式为 Xor。请参考清单 9.16 中的 DitherBtn_Click 事件来详细了解如何在本程序中使用 Xor 模式。

Draw_Blends：本过程使用 Visual Basic 的 Line 语句在 ColorForm 窗体上进行喷涂填色。

Scale：置绘制物体单位为 twip，以确保可绘于窗体之内。

DrawMode=**1**：置笔与填充色为黑色实线。

接下来的 For...Next 循环绘出了五个相互覆盖的圆。

```
For r=0 To 5000 step 1000
Circle (1000+r,Height-1200),600
Next r
```

DrawMode=7：重置绘制模式为 Xor 模式

Draw_Blends：再次以 Visual Basic 的 Line 语句进行喷涂填色。

DrawMode=13：喷涂填充完毕，重新设成最初的绘制模式。

9.18　本章小结

本章提供了几种选择颜色的基本方法，在应用中可以使用它们，第一种方法即 RGBColor&，比较适用于抖色或者 Visual Basic Pelette 文件。这一方法没有产生一个用户调色板，而是使用当前系统已有的调色板或者是 Visual Basic 的调色板。在本程序中，实现一个 RGB 喷涂填色的顺序如下：

1. PicRGB_Paint 事件在启动时绘制二个 121 色的调色板，使用的是 Visual Basic 的 RGB 语句。
2. RGB_MouseDown 事件找到从 ColorIndex 数组中选择的颜色，并用十六进制或 Mod

分解方案来提取其红、绿、蓝分量的值。

3. Fill_Button_Click 事件(喷涂填色按钮)开始用 Visual Basic 的 Line 语句进行喷涂填色，通过 RGBcolor& 过程来填充每一颜色。

第二种方法是 APIcolor& 方法，该方法比较适于纯色的填充(256 色及以上)或自定义的调色板，这一方法需要使用 Type 结构及 API 调色板函数。

本程序中，实现一个 API 喷涂填充的顺序如下：

1. 由 PicRGB_Paint 用 Visual Basic 的 RGB 语句画出 121 色的两个调色板。
2. RGB_MouseDown 事件找到从 ColorIndex 数组中选择的颜色，并用十六进制或 Mod 分解方案来提取其红、绿、蓝分量的值。
3. DitherBtn_Click API 抖色按钮调用 Draw_Blends 过程。
4. Draw_Blends 调用 CreateCustomPalette 过程，该过程使用 LOGPALETTE 及 PALETTEENTRY 结构来形成一个调色板数组——CustomPalette。
5. Draw_Blends 接下来使用 Line 语句来实现喷涂填色，其颜色填充由 APIcolor& 实现。

APIcolor& 与 RGBcolor& 方法的效果基本相同，只不过 APIcolor& 在 CreateCustomPalette 过程中用 API 调色板函数生成一个新的自定义调色板数组，该数组的颜色值存放于早先定义好的 ColorIndex 数组中。

当以结构数组的方式定义好用户调色板后，可以将其置于剪贴板上，通过剪贴板可以直接将调色板(纯色的而非抖色)粘贴到窗体中。

第10章　Bézier 曲线程序

Mini-Bézier 程序是绘制 Bézier 曲线的一个独立应用程序。它给出了绘制 Bézier 曲线所需要的基本结构。Mini-Bézier 程序中绘制 Bézier 曲线的过程与第 3 章的 Artison 程序及第 15 章的 ArtAPI 程序完全相同，包括用来建立 Bézier 曲线的变量名也和第 2 章给出的 Artison 程序的 Declaration 部分一模一样，Mini-Bézier 程序用于试验产生 Bézier 曲线的各种不同方法。如果使用的是 Windows 的 32 位操作系统，就可以使用新的 GDI32 库，GDI32 库内有一大批 API 绘图函数可以应用于 Visual Basic 的 32 位平台，在这些新的 GDI32 API 函数中，用来产生 Bézier 曲线的函数有两个：**PolyBézier**和 PolyBézierTo。

只要提供必需的 4 个曲线点就可以使用 PolyBézier 函数产生一条 Bézier 曲线，四个曲线点是指两个图柄(handle)和曲线的两个节点(node)，如第 2 章的图示所给出，PolyBézierTo 函数可以将一条曲线的节点与另一对象或坐标点相连接。

当使用 API PolyBézier 函数时，将不再使用 BezDraw 或 BezDivide 子程序(见清单 10.2)来计算 deCasteljau 分割公式以画一条曲线。PolyBézier 函数不产生 Bézier 曲线的图柄、lever 或节点，任何改变节点、图柄的事件都可以最小化并且只需将 BezDraw 变为一个语句——PolyBézier。

有一些示例程序包允许通过拖拉鼠标来控制 Bézier 曲线，这种方法自动地移动 Bézier 曲线的图柄以适应曲线的新的坐标位置，Mini-Bézier 程序是实验曲线及节点编辑特征的良好工具。

Mini-Bézier 程序窗体和控制对象的属性如表 10.1 所示。

表 10.1　Mini-Bézier 程序窗体和控制对象的属性

Object:Form			
Object Name:Form1			
BackColor	&H0000FFFF&	Caption	"Bézier Curves"
Height	4830	Left	180
Top	765	Width	6180
Object:Command Button			
Object Name:Exit			
Caption	"Exit"	Height	375
Left	135	Top	3960
Width	1050		
Object:Textbox			
Object Name:Text1			
Height	375	Left	1260
Top	3960	Width	915
Object:Textbox			
Object Name:Text2			
Height	375	Left	2205
Top	3960	Width	915

Object:Command Button

Object Name:ClearBtn

Caption	"Clear"	Height	375
Left	3195	Top	3960
Width	1215		

Object:Picture Box

Object Name:Picture1

DrawMode	6' Invert	Height	3795
Left	135	Top	90
Width	5835		

Object:Picture Box

Object Name:Node1

BackColor	&H00C00000&	BorderStyle	0 'None
Height	105	Left	0
ScaleHeight	105	ScaleWidth	105
Top	495	Width	105

Object:Picture Box

Object Name:Node2

BackColor	&H00C00000&	BorderStyle	0 'None
Height	105	Left	0
ScaleHeight	105	ScaleWidth	105
Top	720	Width	105

Object:Picture Box

Object Name:Handle1

BackColor	&H00FF00FF&	BorderStyle	0 'None
DrawMode	6 'Invert	Height	105
Left	0	ScaleHeight	105
ScaleWidth	105	Top	0
Width	105		

Object:Picture Box

Object Name:Handle2

BackColor	&H00FF00FF&	BorderStyle	0 'None
DrawMode	6 'Invert	Height	105
Left	0	ScaleHeight	105
ScaleWidth	105	Top	240
Width	105		

Object:Picture Box

Object Name:Shape1

Height	105	Left	0
Top	1200	Width	105

Object:Shape

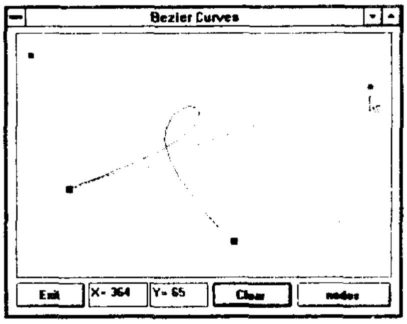
Object Name:Shape2

Height	105	Left	0
Top	1320	Width	105

Object:Command Button

Object Name:Nodehandle

Caption	"nodes"	Height	375
Left	4500	Top	3960
Width	1455		



10.1　16 位 Bézier 曲线程序使用的变量名

清单 10.1 给出的 Mini-Bézier 程序建立了各变量，这些变量将用于 deCasteljau 公式以预绘出 Bézier 曲线上的点，当使用的操作系统为 32 位，并且使用了 PolyBézier 函数时，清单中所列的 D1,D2,D3,E1,E2 和 C1 数组变量将不需要。

清单 10.1　启动时设置的初始变量

```
Declaration
Declare Function Polyline Lib "GDI" (ByVal hDC%, P As POINTS,
    ByVal NUM_PTS%) As Integer
Const Limit = 1000
Const MAX_POINTS = 33
Dim P_Index(1 To MAX_POINTS) As POINTS
Dim P As Integer
Dim X1 As Integer
Dim Y1 As Integer
Dim X2 As Integer
Dim Y2 As Integer
Dim P1 As POINTS
Dim H1 As POINTS
Dim H2 As POINTS
Dim P2 As POINTS
Dim D1(1 To MAX_POINTS) As POINTS
Dim D2(1 To MAX_POINTS) As POINTS
Dim D3(1 To MAX_POINTS) As POINTS
Dim E1(1 To MAX_POINTS) As POINTS
Dim E2(1 To MAX_POINTS) As POINTS
Dim C1(1 To MAX_POINTS) As POINTS
Const Yellow = &HFFFF&
Dim PixelX As Integer
Dim PixelY As Integer
```

Declare Function Polyline：API 函数中的画线函数，与 Visual Basic 的 Line 语句相同。

Const MAX_POINTS：存储 Bézier 曲线上点的数目。

Dim P_Index(1 To MAX_POINTS)AS POINTS：存储 Bézier 曲线上点的索引，通过 POINTS 结构用一个数组名存储每一点的 X、Y 坐标值。

Dim P AS Integer：索引的起始点。

X1,X2,Y1,Y2：当前的屏幕坐标值，当将某一图形对象移至一个新位置时，用来清除上

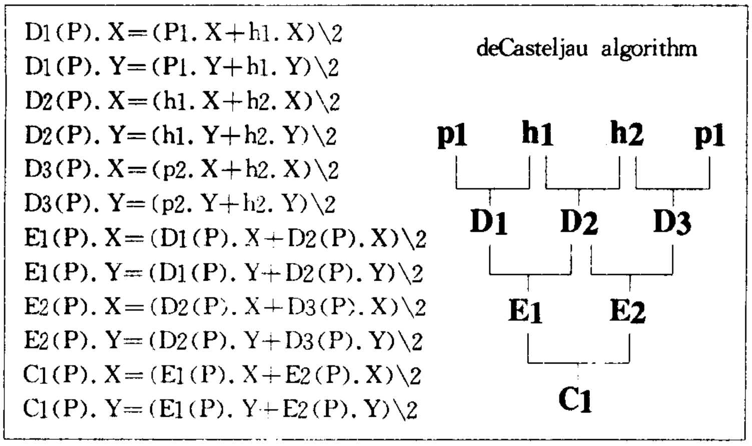
一位置的图形对象。

P1,H1,H2,P2:Bézier 曲线的四个控制点。P1=Bézier node Point1,H1=Bézier Handle1,P2=Bézier node Point2,H2=Bézier Handle2。

D1,D2,D3,E1,E2,C1:Bézier 曲线的分割点。经过 BezDivide 中的 deCasteljau 分割公式的计算,变量 C1 表示曲线上的一个分割点。

Const Yellow:黄颜色的十六进制码,当移动一条曲线的图柄时,点线的颜色为青色,当在 Xor 模式下用 Draw/Node 绘制时,象素点的颜色由黄变青,Xor 模式在线移动时存储了屏幕的背景色。

图 10.1 给出了公式的结构(清单 10.1 创建了此公式)。



D1(P).X=(P1.X+h1.X)\2
D1(P).Y=(P1.Y+h1.Y)\2
D2(P).X=(h1.X+h2.X)\2
D2(P).Y=(h1.Y+h2.Y)\2
D3(P).X=(p2.X+h2.X)\2
D3(P).Y=(p2.Y+h2.Y)\2
E1(P).X=(D1(P).X+D2(P).X)\2
E1(P).Y=(D1(P).Y+D2(P).Y)\2
E2(P).X=(D2(P).X+D3(P).X)\2
E2(P).Y=(D2(P).Y+D3(P).Y)\2
C1(P).X=(E1(P).X+E2(P).X)\2
C1(P).Y=(E1(P).Y+E2(P).Y)\2

图 10.1　deCasteljau 公式的结构

10.2　计算曲线上的点

你可以通过 deCasteljau 简单地计算出 Bézier 曲线上的点。只要简单地对 Bézier 曲线的四个控制点(P1,P2,H1,H2)进行二分法分割,最终就可以得到曲线上一点的坐标。

清单 10.2　deCasteljau 分割公式

```
Sub BezDivide (P1 As POINTS, H1 As POINTS, H2 As POINTS, P2 As POINTS)
D1(P).X = (P1.X \ 2 + H1.X \ 2)
D1(P).Y = (P1.Y \ 2 + H1.Y \ 2)
D2(P).X = (H1.X \ 2 + H2.X \ 2)
D2(P).Y = (H1.Y \ 2 + H2.Y \ 2)
D3(P).X = (P2.X \ 2 + H2.X \ 2)
D3(P).Y = (P2.Y \ 2 + H2.Y \ 2)
E1(P).X = (D1(P).X \ 2 + D2(P).X \ 2)
E1(P).Y = (D1(P).Y \ 2 + D2(P).Y \ 2)
E2(P).X = (D2(P).X \ 2 + D3(P).X \ 2)
E2(P).Y = (D2(P).Y \ 2 + D3(P).Y \ 2)
C1(P).X = (E1(P).X \ 2 + E2(P).X \ 2)
C1(P).Y = (E1(P).Y \ 2 + E2(P).Y \ 2)
'Picture1.DrawWidth = 4: Picture1.DrawMode = 6
'Picture1.PSet (C1(P).X, C1(P).Y)
'Picture1.DrawWidth = 1
```

```
End Sub
```

Di(P).X＝(P1.X\2＋H1.X\2)：本语句中使用了整除，目的是为了使结果为整型，你可以将语句(P1.X\2＋H1.X\2)缩短写为(P1.X＋H1.X)\2。如果是在 Pixel 模式下，这两种写法结果相同，在 twips 下，P1.X＋H1.X 的值在整型数的范围之内——(－32768 到 32767)，你可以通过 PSet 语句画出曲线上的点(C1)，可以查看第 2 章的 Artisan 程序来了解画一条曲线的具体内容。

注释： 使用 deCasteljau 公式时，"\"除号为整型数除法，"/"除号使用于浮点数除法，如果使用"/"除号，会大大降低运行速度。

10.3　绘出曲线上的点

BezDraw 子程序(见清单 10.2)中的变量 P 给出索引值；该索引存放着由 deCasteljau 计算出的每一个值，变量 P 也给出组成曲线的点的号码，从曲线的起始点 P1 开始到曲线的终点 P2，点的号码为从 2 到 32，所以，第 17 号点位于曲线的中央。

如清单 10.3 所示，首先画出曲线的左半部分，计算每一点时，都采用同样的分割公式，这样，首先画出第 2 到第 16 点，接着画曲线的右半部分，画出第 18 到第 32 点。右半部分与左半部分成镜象对称。

清单 10.3　画出曲线上的每一点

```
Sub BezDraw ()
P = 17: BezDivide P1, H1, H2, P2
Leftside
P = 13: BezDivide P1, D1(17), E1(17), C1(17)
P = 9: BezDivide P1, D1(13), E1(13), C1(13)
P = 5: BezDivide P1, D1(9), E1(9), C1(9)
P = 3: BezDivide P1, D1(5), E1(5), C1(5)
P = 2: BezDivide P1, D1(3), E1(3), C1(3)

P = 4: BezDivide C1(3), E2(3), D3(3), C1(5)
P = 7: BezDivide C1(5), E2(5), D3(5), C1(9)
P = 6: BezDivide C1(7), D1(7), E1(7), C1(5)
P = 8: BezDivide C1(7), E2(7), D3(7), C1(9)
P = 11: BezDivide C1(9), E2(9), D3(9), C1(13)
P = 10: BezDivide C1(11), D1(11), E1(11), C1(9)
P = 12: BezDivide C1(11), E2(11), D3(11), C1(13)
P = 15: BezDivide C1(13), E2(13), D3(13), C1(17)
P = 14: BezDivide C1(15), D1(15), E1(15), C1(13)
P = 16: BezDivide C1(15), E2(15), D3(15), C1(17)
Rightside
P = 21: BezDivide C1(17), E2(17), D3(17), P2
P = 25: BezDivide C1(21), E2(21), D3(21), P2
P = 29: BezDivide C1(25), E2(25), D3(25), P2
P = 31: BezDivide C1(29), E2(29), D3(29), P2
P = 32: BezDivide C1(31), E2(31), D3(31), P2
P = 30: BezDivide C1(29), D1(31), E1(31), C1(31)
P = 27: BezDivide C1(25), D1(29), E1(29), C1(29)
P = 28: BezDivide C1(29), E2(27), D3(27), C1(27)
```

```
P = 26：BezDivide C1(25)，D1(27)，E1(27)，C1(27)
P = 23：BezDivide C1(21)，D1(25)，E1(25)，C1(25)
P = 24：BezDivide C1(25)，E2(23)，D3(23)，C1(23)
P = 22：BezDivide C1(21)，D1(23)，E1(23)，C1(23)
P = 19：BezDivide C1(17)，D1(21)，E1(21)，C1(21)
P = 20：BezDivide C1(21)，E2(19)，D3(19)，C1(19)
P = 18：BezDivide C1(17)，D1(19)，E1(19)，C1(19)
''''''''''''''''''''''''''''''''''''''''''''''''

Picture1.DrawMode = 6：Picture1.DrawStyle = 0
    P_Index(1).X = P1.X \ PixelX
    P_Index(1).Y = P1.Y \ PixelY
For P = 2 To 32
    P_Index(P).X = C1(P).X \ PixelX
    P_Index(P).Y = C1(P).Y \ PixelY
    Next P
P_Index(33).X = P2.X \ PixelX
P_Index(33).Y = P2.Y \ PixelY
NUM_PTS = 33
R = Polyline(Picture1.hDC，P_Index(1)，NUM_PTS)
End Sub
```

Left Side：P 是一整数值，表示数组 D1，D2，D3，E1，E2 和 C1 的索引号，P 也表示了曲线上点的顺序，从 BezNode1(P1)点开始增加到 32 个点，最后再与第 33 点——BezNode2(P2)相连。

传递给 BézierDivide 函数的四个变量为 Bézier 曲线的节点 P1，P2 以及图柄 H1，H2。如图 10.2 所示，第一块的六个计算公式将左半部分曲线分割，这些分割点被存放入 C1 数组索引号中，因此，最开始的六个点存放于 C1(17)，C1(13)，C1(9)，C1(5)，C1(2)中。

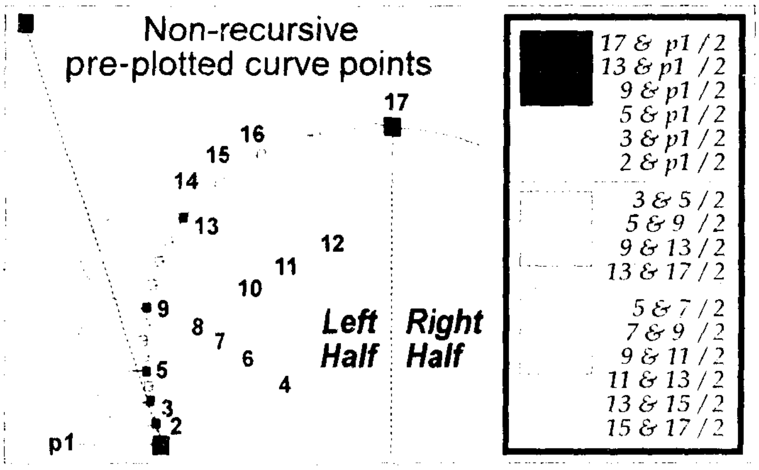

Non-recursive
pre-plotted curve points
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
p1
Left Half
Right Half
17 & p1 /2
13 & p1 /2
9 & p1 /2
5 & p1 /2
3 & p1 /2
2 & p1 /2
3 & 5 /2
5 & 9 /2
9 & 13 /2
13 & 17 /2
5 & 7 /2
7 & 9 /2
9 & 11 /2
11 & 13 /2
13 & 15 /2
15 & 17 /2

图 10.2　左半部分曲线上画点的顺序

接着要设置的 10 个点只需置于已有的六个点的中点处即可，如第 3 与第 5 点之间，置一灰色点 C(4)。注意，如果一个灰色点与一个黑色点之间有视觉可觉察到的空隙，这两点之间也应再设置一灰色点，如 C(6)即为 C(7)与 C(5)之中点。

从 P=4 开始的第二块程序，可以注意到第一个变量与第四个变量，均为 C1——作为

Bézier 曲线的两个节点；中间的两个变量作为 Bézier 曲线的图柄控制对象，根据这段程序，产生了左半部分曲线的各个点，从 P1 到 C(17)，以顺时针顺序得到了结果；对于右半部分曲线，计算是相同的，只不过是逆时针顺序而已。P1 是左半部分的起点，而 P2 是右半部分的终点。

可以控制 Bézier 曲线的 ScaleMode 有 Twip 或 Pixel 两种模式，当以 Pixel 模式绘图时，在 Bézier 曲线的节点(Nodel 和 Node2)附近，曲线会有一些弯曲，这是由于 deCasteljau 分割公式的误差以及 C1 点过于靠近节点造成的，清单 10.4 给出 Form_Load 事件的程序，该程序使用 TwipPerPixel 函数将 twip 单位换为 Pixel 单位。

清单 10.4　启动时的初始值

```
Sub Form_Load ()
    PixelX = Screen.TwipsPerPixelX
    PixelY = Screen.TwipsPerPixelY
End Sub
```

PixelX,PixelY：将 Twips 单位换算为 Pixel 单位。进行这种内部的换算有两个原因：(1) API PolyLine 函数只接收 X,Y 的值为 Pixel 坐标。(2) 当移动鼠标时，Text1，Text2 两个文本框显示的 X,Y 值为 Pixel 坐标。

清单 10.5 给出 Form_Resize 事件，该事件将窗体置于屏幕中心。

清单 10.5　将主窗体置于屏幕中心

```
Sub Form_Resize ()
    Form1.Left = (Screen.Width - Form1.Width) \ 2
    Form1.Top = (Screen.Height - Form1.Height) \ 2
End Sub
```

Form1.Left(Top)：使窗体位于屏幕中心。

清单 10.6 给出 Picturel-MouseDown 事件的程序，该事件清除线或图形并将节点值重置为初始值，该事件使用了 Line_MouseDown 子程序库，这段程序与第 3 章的 Artisan 程序相同。

清单 10.6　Picturel_MouseDown 事件

```
Sub Picture1_MouseDown (Button%, Shift%, X!, Y!)
    ClearBtn_Click
    Line_MouseDown X, Y
End Sub
```

ClearBtn：触发窗体上的 Clear 按钮以清屏。

Line_MouseDown：调用本过程来重置线的起始点，本语句也可以换上更复杂的程序段，以完成其他应用。

清单 10.7 给出 Picturel_MouseMove 事件的程序，当鼠标指针在图片框内移动时，它跟踪显示鼠标指针的位置。

清单 10.7　鼠标在窗体内移动

```
Sub Picture1_MouseMove (Button%, Shift%, X!, Y!)
    Text1.Text = "X= " + Format(X) \ PixelX
    Text2.Text = "Y= " + Format(Y) \ PixelY
    Line_MouseMove Button, X, Y
End Sub
```

Text1.Text，Text2.Text：显示当前鼠标位置的 X、Y 值。

Line_MouseMove：调用子程序，画出一条直线至鼠标位置。

清单 10.8 给出 Picturel_MouseUp 事件的程序，该程序调用 Line_MouseUp 过程计算出线的最终坐标值。

清单 10.8　Picturel_MouseUp 事件

```
Sub Picture1_MouseUp (Button%, Shift%, X!, Y!)
    Line_MouseUp X, Y
End Sub
```

Line_MouseUp：调用终止画线过程。

图 10.3 给出一条线怎样转换为一条曲线。

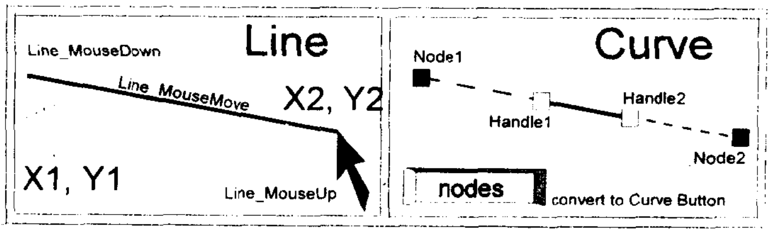

Line_MouseDown
Line
Line_MouseMove
X2, Y2
X1, Y1
Line_MouseUp
Node1
Curve
Handle2
Handle1
Node2
nodes
convert to Curve Button

图 10.3　一条线转换为一条曲线

10.4　在窗体内画一条线

首先在 Picturel 图片框控制对象内画一条线，以后可以将此线转换为曲线，画线的过程很简单，只需在线的起点处按下鼠标，拖鼠标至线的终点，松开鼠标按钮则线画完。

清单 10.9 的 Line_MouseDown 事件程序给出初始化一条线的起点坐标。

清单 10.9　设置一条线的起始点

```
Sub Line_MouseDown (X As Single, Y As Single)
    X1 = X:Y1 = Y
    X2 = X1:Y2 = Y1
End Sub
```

X1,Y1,Y2,Y2:置所有画线点位置为当前的 X,Y 值。

清单 10.10 给出 Line_MouseMove 事件的程序,该程序在拖动鼠标时画线。

清单 10.10 画线

```
Sub Line_MouseMove (Button As Integer, X!, Y!)
    If Button Then
        Picturel.Line (X1, Y1)-(X2, Y2), QBColor(4)
            X2 = X
            Y2 = Y
        Picturel.Line (X1, Y1)-(X2, Y2), QBColor(4)
    End If
End Sub
```

If Button Then:如果鼠标被按下,则继续执行。

Picturel. Line:画线。

X2,Y2:置 X2,Y2 为当前的鼠标位置值,画线时采用的是 Inverse 模式,在该模式下绘制图象的颜色与背景色反色(黑色背景上图象为白色),采用本模式,因为重绘两次则原有线条消失,故可以产生移动的效果。

Picturel1. Line:重绘线条以清除刚画过的线。

清单 10.11 给出 Line_MouseUp 事件的程序,该程序得到线的坐标值,当鼠标被松开时,两个节点即在线的两个端点处定位,两个节点的 X,Y 坐标值即被存入变量 P1 和 P2 中,P1 存放线的起点坐标,P2 存放线的终点坐标。

清单 10.11 设置直线的节点位置

```
Sub Line_MouseUp (X As Single, Y As Single)
    If X1 = X Or Y1 = Y Then Exit Sub
        Nodel.Move X1 - 45, Y1 - 45
        Node2.Move X - 45, Y - 45
        Nodel.ScaleLeft = Nodel.Left
        Nodel.ScaleTop = Nodel.Top
        Node2.ScaleLeft = Node2.Left
        Node2.ScaleTop = Node2.Top
    P1.X = X1
    P1.Y = Y1
    P2.X = X2
    P2.Y = Y2
End Sub
```

If X1=X or Y1=Y:线的长度小于 1 个象素,则退出本进程。

Node(i),Move:重定位两个节点。

Node1(i). ScaleLeft(Top):得到两个节点在窗体中的内部坐标值。

P1. X,P1. Y,P2. X,P2. Y:在 P1,P2 中放入线的起点、终点坐标值。

图 10.4 给出了窗体及其控制对象的示意图。

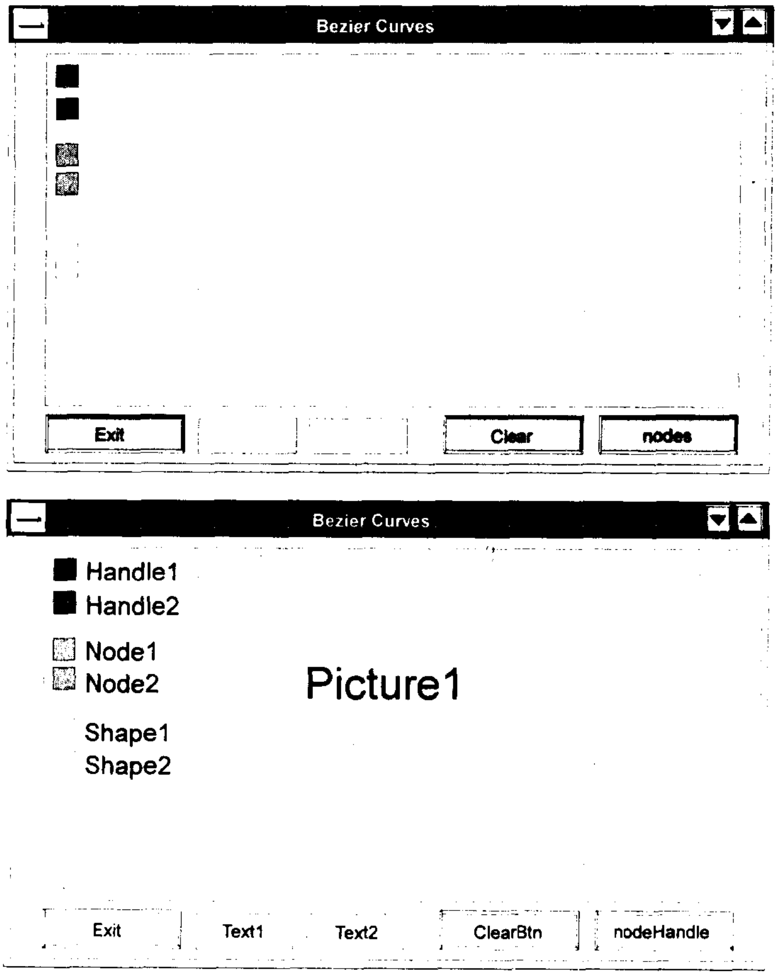

Bezier Curves
Exit
Clear
nodes
Bezier Curves
Handle1
Handle2
Node1
Node2
Picture1
Shape1
Shape2
Exit
Text1
Text2
ClearBtn
nodeHandle

图 10.4　窗体与控制对象

10.5　移动前清除旧的线条

当一个图柄或节点被移动时，调用清单 10.12 的程序来画出一条 Bézier 曲线，在 Mini-Bézier 程序中，不使用 AutoRedraw 来保存曲线图，只需在将一条直线变为曲线时，清除旧的线条即可。

清单 10.12　清除旧的线条

```
Sub BezMove ()
    Picture1.DrawMode = 7；Picture1.DrawStyle = 2
    Picture1.Line (P1.X、P1.Y)－(H1.X，H1.Y)，Yellow
    Picture1.Line (P2.X，P2.Y)－(H2.X，H2.Y)，Yellow
    If P = 0 Then
        Picture1.DrawStyle = 0；Picture1.DrawMode = 13
        Picture1.Line (X1，Y1)－(X2，Y2)，QBColor(15)
        BezDraw
    End If
    Picture1.DrawMode = 7；Picture1.DrawStyle = 2
    Picture1.Line (P1.X，P1.Y)－(H1.X，H1.Y)，Yellow
    Picture1.Line (P2.X，P2.Y)－(H2.X，H2.Y)，Yellow
```

```
End Sub
```

Picture1. DrawMode：点划线将 Bézier 曲线点和 Bézier 图柄连接起来。由于它们可画在最初用 Copy_Pen 模式(13)绘制的曲线的顶上或穿过曲线，必须改变模式。

Picture1. DrawStyle：由于想画的线为点线，应将 DrawStyle 改为 Dot(2)。

Picture1. Line：画点划线。

If P=0：变量 P 为 0 表示该线已被画出，因为要将此线转换为曲线，故应将此线条抹去，清除线条的方法是在黑色图象上再画一条白线，如此即可将原图象抹去。

Picture1. DrawStyle：重置 DrawStyle 为 Solid(0 为正常)。

Picture1. DrawMode：重置为 DrawMode 为 Copy_Pen(13 normal)，以便用白色删去原来的线，还可以用 DrawMode(白色)来达到此效果。

Picture1. Line：在线条上画一白线，以清除原有线条。

BezDraw：调用 BezDraw 过程在线条的位置画一条 Bézier 曲线。

10.6 画一条新曲线前重置各初始值

清单 10.13 给出 ClearBtn_Click 事件的程序，该程序重置各控制对象为初始值，与第 2 章的 Artisan 程序中将节点与图柄隐藏起来不同，在本程序中，只是将这些节点、图柄移到窗体的一侧。

首先，在窗体中间画一条线，接着在窗体内单击鼠标，线条消失了，但是线条的起点、终点坐标值仍被存放着，此时，仍可以将位于窗体一侧的某一节点或图柄移动，这个过程显示了节点、图柄的位置以及它们与一条曲线的关系，它也给出了通过 Node1，Node2 两个图片框控制对象移动曲线的示例。

清单 10.13 画一条新线之前重置各值

```
Sub ClearBtn_Click ()
If Node1.Left = 0 Then Exit Sub
    Picture1.Cls
    Handle1.Move 0, 150
    Handle2.Move 0, 300
    Node1.Move 0, 450
    Node2.Move 0, 600
    Shape1.Move 0, 900
    Shape2.Move 0, 1200
        P = 0
        Picture1.ForeColor = QBColor(1)
        Picture1.DrawMode = 6
        Picture1.DrawStyle = 0
End Sub
```

If Node1. Lef=0 Then：各控制对象已经在初始位置，退出。

Picture1. Cls：清屏。

Handle1. Move：将本控制对象移动回初始位置。

Node1. Move：将本控制对象移动回初始位置。

Shaple. Move：将本控制对象移动回初始位置。

P=0：置清除线的标志变量 P 为 0。

Picture1. DrawMode：置 DrawMode 为 Inverse mode(6)。

Picture1. DrawStyle：置 DrawStyle 为 Solid(0)。

10.7　Bézier 图柄 1

清单 10.14 给出 Handle1_MouseDown 事件的程序，该程序重绘曲线以产生移动的效果。

清单 10.14　用鼠标单击图柄后线的重绘

```
Sub Handle1_MouseDown (Button%, Shift%, X!, Y!)
    BezHandle1. Visible = False
    BezMove
End Sub
```

BezHandle1：置当前的图柄为不可见，节点与图柄均为图片框控制对象，各节点与图柄之间由点划线相连接。当曲线通过移动图柄而移动时，为了控制线重绘时能产生移动的效果，首先应置图柄为不可见。

BezMove：调用 BezMove 过程（见清单 10.12）重画曲线。

10.8　重绘图柄 1 的控制线

控制线是指曲线的端点与图柄之间相连的点划线，当使用一个图柄来移动曲线时，控制线必须被重绘，程序见清单 10.15。

清单 10.15　图柄 1 移动时重绘 Bézier 曲线

```
Sub Handle1_MouseMove (Button %, Shift%, X!, Y!)
Text1. Text = "X= " + Format $ (X)
Text2. Text = "Y= " + Format $ (Y)
    If Button Then
        BezDraw
        Picture1. DrawMode = 7
        Picture1. DrawStyle = 2
        Picture1. Line (P1. X, P1. Y)-(H1. X, H1. Y), Yellow
            H1. X = X
            H1. Y = Y
        Shape1. Left = H1. X - 45
        Shape1. Top = H1. Y - 45
        Picture1. Line (P1. X, P1. Y)-(H1. X, H1. Y), Yellow
    BezDraw
End If
End Sub
```

Text1. Text：显示图片框控制对象的 X 坐标。

Text2. Text：显示图片框控制对象的 Y 坐标。

If Button Then：如果鼠标被按下，则接着处理以下程序。

BezDraw：调用 BezDraw 过程画 Bézier 曲线。

ArtWork. DrawMode：置 DrawMode 为 Xor(7)，这样，当点划线绘成的控制线通过其他图形时，不会将其他图形抹掉。

ArtWork. Line：画点划线，在 Xor 模式下，黄色的控制线将变为青色。

H1. X=X：置控制线的端点位置为鼠标的 X 坐标。如果在 Xor 模式下绘图，当重画控制线时，它经过的图形都将被保存下来，这将产生移动的效果。

H1. Y=Y：同 H1. X=X，不过此处设置的是鼠标的 Y 坐标。

Shapel. Left,Shapl. Top：用 Shape 控制对象取代 Handle1 图片框控制对象。如果 Left 和 Top 属性移动 Handle1 图片框，该图片框将来回跳动，就像是手工绘制过图柄似的。

ArtWork. Line：重画一遍点划线，以保证控制线移动经过的线条不被破坏。要产生移动的效果，设置绘图模式为 Xor(7)，画两遍物体，因为 Xor 模式下背景图象将被保存。

BezDraw：重绘整条曲线以产生移动的效果。

10.9 重设图柄 1 的大小

当 Bézier 曲线的图柄被移动到一个新的位置时，Handle1 图片框控制对象将被重画，因为它位于 Picture1 图片框内，所以大小将由 Picture1 图片框的内部坐标决定(见清单 10.16)。

清单 10.16 重画 Bézier 曲线的图柄大小

```
Sub Handle1_MouseUp (Button%, Shift%, X!, Y!)
    Handle1. Move X — 45, Y -- 45
    Handle1. ScaleLeft = Handle1. Left
    Handle1. ScaleTop = Handle1. Top
    Handle1. Visible = True
End Sub
```

Handle1. Move：移动图柄，使其与点线相连。

Handle1. ScaleLeft(Top)：以内部坐标系重置图柄的大小。

Handle1. Visible：重定位图柄后置其可见。

10.10 Bézier 图柄 2

清单 10.17 给出 Handle2_MouseDown 事件，该程序重绘曲线时可以产生移动的效果。

清单 10.17 单击 Handle2 后线的重绘

```
Sub Handle2_MouseDown (Button%, Shift%, X!, Y!)
    BezHandle2. Visible = False
    BezMove
End Sub
```

关于怎样构造上述代码的细节，请参见清单 10.14。

10.11　Handle2 的控制线

控制线是指曲线的端点与图柄之间相连的点划线，当使用一个图柄来移动曲线时，控制线必须重画，程序如清单 10.18 所示。

清单 10.18　移动 Handle2 后线的重绘

```
Sub Handle2_MouseMove (Button %, Shift%, X!, Y!)
Text1.Text = "X= " + Format $ (X)
Text2.Text = "Y= " + Format $ (Y)
    If Button Then
        BezDraw
        Picture1.DrawMode = 7
        Picture1.DrawStyle = 2
        Picture1.Line (P1.X, P1.Y)-(H1.X, H1.Y), Yellow
             H2.X = X
             H2.Y = Y
        Shape2.Left = H1.X - 45
        Shape2.Top = H1.Y - 45
        Picture1.Line (P2.X, P2.Y)-(H2.X, H2.Y), Yellow
    · BezDraw
End If
End Sub
```

关于怎样构造上述代码的细节，请参见清单 10.15。

10.12　重设图柄 2 的大小

移动图柄后，必须改变 Handle2 图片框的大小，因为它位于 Picture1 图片框内，大小将由 Picture1 图片框的内部坐标决定，程序如清单 10.19 所示。

清单 10.19　重设 Bézier 曲线的图柄大小

```
Sub Handle2_MouseUp (Button%, Shift%, X!, Y!)
    Handle2.Move X - 45, Y - 45
    Handle2.ScaleLeft = Handle2.Left
    Handle2.ScaleTop = Handle2.Top
    Handle2.Visible = True
End Sub
```

关于怎样构造上述代码的细节，请参见清单 10.16。

10.13　通过节点移动一条曲线

位于曲线两端的两个节点分别为 Node1 和 Node2。可以通过这两个节点来移动曲线。

在不移动 Handle1 的情况下，移动曲线时的 Node1 鼠标事件是以 Handle 子程序（见清单 10.14）为基础的，清单 10.20 给出了曲线的设置。

清单 10.20 Node1_MouseDown 事件

```
Sub Node1_MouseDown (Button%, Shift%, X!, Y!)
    If P = 0 Then Nodehandle_Click
        BezMove
        Picture1.Line (P1.X, P1.Y)-(H1.X, H1.Y), Yellow
        Node1.Visible = False
        Handle1.Visible = False
End Sub
```

If P=0 Then Nodehandle_Click：线画完之后，在线的两端节点也立即被画出，但因为图柄和节点只有在 Nodes 按钮(即 Nodehandle_Click)被按下后才显示出来，故变量 P 用来作为一个标志来触发 Nodes 按钮。

BezMove：画第一条 Bézier 曲线，同时抹去当前显示的直线。

Picture1.Line：画出控制线。

Node1.Visible：移动时置 Node1 为不可见。

Handle1.Visible：置 Handle1 为不可见。

当用鼠标指针移动第一个节点时，清单 10.21 给出的 Node_MouseMove 事件画出一个圆。

清单 10.21 当 Node1 被移动后，Bézier 曲线的重绘

```
Sub Node2_MouseMove (Button%, Shift%, X!, Y!)
Text1.Text = "X= " + Format$(X)
Text2.Text = "Y= " + Format$(Y)
    If Button Then
        BezDraw
        P1.X = X
        P1.Y = Y
        H1.X = (X - X1) + Handle1.Left + 45
        H1.Y = (Y - Y1) + Handle1.Top + 45
        Shape1.Left = P1.X - 45
        Shape1.Top = P1.Y - 45
        BezDraw
    End If
End Sub
```

Text1.Text：显示图片控制对象的 X 坐标。

Text2.Text：显示图片控制对象的 Y 坐标。

If Button Then：如果鼠标被按下，则接着执行以下程序。

BezDraw：调用 BezDraw 子程序(见清单 10.3)画第一条 Bézier 曲线。

P1.X=X：置控制线与曲线的终点为鼠标的当前坐标值，如果在 Xor 模式下绘制，第二次重画时控制线经过的图象均被保存，这样才能产生移动的效果。

P1.Y=Y：同 P1.X=X，只是此时设置的是 Y 坐标。

H1.X、H1.Y：移动曲线的图柄，保持初始值时点与图柄的距离与角度。

Shape1. Left，Shape1. Top：用 Shape 控制对象取代 Node1 图片框控制对象。如果用 Left 和 Top 属性来移动 Node1 图片框，该图片框将来回跳动，就像是手工绘制过节点似的。

BezDraw：重绘整条曲线以产生运动的效果。

10.14 重绘节点 1

当曲线的节点和图柄被移动到一个新的位置时，必须重设这两个图片框控制对象的比例，Node1 图片框控制对象的大小依赖于 Picture1 图片框的内部坐标系（见清单 10.22）。

清单 10.22 重设 Node1 图片框控制对象的比例

```
Sub Node2_MouseUp (Button%, Shift%, X!, Y!)
Node1.Move X - 45, Y - 45
Node1.ScaleLeft = Node1.Left
Node1.ScaleTop = Node1.Top
Node1.Visible = True
    Handle1.Move (X - X1) + Handle1.Left, (Y - Y1) + Handle1.Top
    Handle1.ScaleLeft = Handle1.Left
    Handle1.ScaleTop = Handle1.Top
    Handle1.Visible = True
X1 = P1.X
Y1 = P1.Y
    Picture1.DrawMode = 7: Picture1.DrawStyle = 2
    Picture1.Line (P1.X, P1.Y)-(H1.X, H1.Y), Yellow
End Sub
```

Node1. Move：将 Node1 移到新的位置。

Node1. ScaleLeft(Top)：以 Picture1 的内部坐标重置节点的大小。

Node1. Visible：置 Node1 图片框为可见。

Handle1. Move：重新定位 Handle1。

Handle1. ScaleLeft(Top)：以 Picture1 的内部坐标重设 Handle1。

Handle1. Visible：置 Handle1 图片框为可见。

X1、1：存放曲线的终点坐标。

Picture1. Line：重画 Node1 与 Handle1 之间的控制线。

10.15 通过 Node2 移动曲线

Node2 的处理与 Node1 的处理完全相同，清单 10.23，10.24，10.25 分别给出了 Node 2_MouseDown，Node2_MouseMove，Node2_MouseUp 事件的程序，可以看到，这些程序与清单 10.20，10.21，10.22 完全一样，在此不再详述。

清单 10.23 Node2_MouseDown 事件

```
Sub Node2_MouseDown (Button%, Shift%, X!, Y!)
    If P = 0 Then Nodehandle_Click
        BezMove
    Picture1.Line (P1.X, P1.Y)-(H1.X, H1.Y), Yellow
    Node1.Visible = False
    Handle1.Visible = False
End Sub
```

关于怎样构造上述代码的细节请参见 Node1_MouseDown 事件(见清单 10.20)。

清单 10.24 当 Node2 移动时重绘 Bézier 曲线

```
Sub Node2_MouseMove (Button%, Shift%, X!, Y!)
Text1.Text = "X= " + Format$(X)
Text2.Text = "Y= " + Format$(Y)
    If Button Then
          BezDraw
      Picture1.DrawMode=7
          P2.X = X
            P2.Y = Y
      H2.X = (X - X2) + Handle2.Left + 45
      H2.Y = (Y - Y2) + Handle2.Top + 45
      Shape2.Left = P2.X - 45
      Shape2.Top = P2.Y - 45
          BezDraw
    End If
End Sub
```

关于怎样构造上述代码的细节请参见 Node1_MouseMove 事件(清单 10.21)。

清单 10.25 重设 Node2 图片框控制对象的比例

```
Sub Node2_MouseUp (Button%, Shift%, X!, Y!)
Node2.Move X - 45, Y - 45
Node2.ScaleLeft = Node2.Left
Node2.ScaleTop = Node2.Top
Node2.Visible = True
    Handle2.Move (X - X2) + Handle2.Left, (Y - Y2) + Handle2.Top
    Handle2.ScaleLeft = Handle2.Left
    Handle2.ScaleTop = Handle2.Top
    Handle2.Visible = True
X2 = P2.X
Y2 = P2.Y
    Picture1.DrawMode = 7: Picture1.DrawStyle = 2
    Picture1.Line (P2.X, P2.Y)-(H2.X, H2.Y), Yellow
End Sub
```

关于怎样构造上述代码的细节请参见 Node1_MouseUp 事件(清单 10.22)。

10.16 将一条直线转换为一条曲线

在窗体中画一条直线以后，单击 Nodes 按钮，将把一条直线转换为一条曲线，NodeHandle_Click 事件在线中距两个端点各 1/4 长度处放置两个图柄，并且将节点与图柄用点划线连接起来，见清单 10.26。

清单 10.26　在直线上设置图柄

```
Sub Nodehandle_Click ()
If Node1.Left = 0 Then Exit Sub
    Picture1.Enabled = True
    Picture1.DrawMode = 7: Picture1.DrawStyle = 2
        MoveLeft = (X2 + X1) \ 2
        MoveTop = (Y2 + Y1) \ 2
    Handle1.Left = (X1 + MoveLeft) \ 2 - 45
    Handle1.Top = (Y1 + MoveTop) \ 2 - 45
    Handle2.Left = (X2 + MoveLeft) \ 2 - 45
    Handle2.Top = (Y2 + MoveTop) \ 2 - 45
    Handle1.ScaleLeft = Handle1.Left
    Handle1.ScaleTop = Handle1.Top
    Handle2.ScaleLeft = Handle2.Left
    Handle2.ScaleTop = Handle2.Top
        H1.X = Handle1.Left + 45
        H1.Y = Handle1.Top + 45
        H2.X = Handle2.Left + 45
        H2.Y = Handle2.Top + 45
    Picture1.Line (X1, Y1)-(H1.X, H1.Y), Yellow
    Picture1.Line (X2, Y2)-(H2.X, H2.Y), Yellow
End Sub
```

If Node1.Left=0 Then：如果节点位于坐标 0 处，说明还未画线，退出本事件。

MoveLeft.MoveTop：代表线的中点的变量。

Handle1(2),Left(Top)：每一个图柄距离一个节点的长度为线段总长的四分之一，−45 为图柄图片框控制对象的中心坐标（以 twip 为单位）。

Handle1(2),ScaleLeft(Top)：调整窗体内控制对象的比例。

H1.X=Handle1.Left+45：置曲线图柄的坐标值。

Picture1.Line：画出节点与图柄之间的控制线。

10.17　本章小结

利用数学公式对曲线的起点、终点以及两个图柄点进行计算，可以产生一个四点的 Bézier 曲线，两个图柄分别控制曲线的左半部或右半部的形状，如果在 16 位的 Windows 操作系统下设计 Bézier 曲线，就需要一个程序来计算并画出曲线。如果工作在 32 位的 Windows 操作系统下，就可以使用 API PolyBézier 函数来计算。

除了本章中介绍四点分割的 Bézier 曲线之外，另一种比较常见的 Bézier 曲线是三点分割的，常用于产生文本字符的轮廓，三点的 Bézier 曲线与四点的 Bézier 曲线基本相同，只是计算上略有不同，而且三点的 Bézier 曲线只使用一个图柄来控制曲线的形状，32 位 API Bézier 函数使用四点 Bézier 曲线的方法。

可以查看第 3 章中建立 Bézier 曲线的详细方法。

第 11 章　绘制圆角程序

绘制圆角(Rounded Corners)程序提供了使用 Line 和 Circle 语句画出圆角矩形的三种方法。这三种方法的选项分别是 CurrentXY,Basic 和 Stretcher,在绘制圆角矩形时,每种方法都略有不同。图 11.1 给出了绘制圆角程序的屏幕显示。

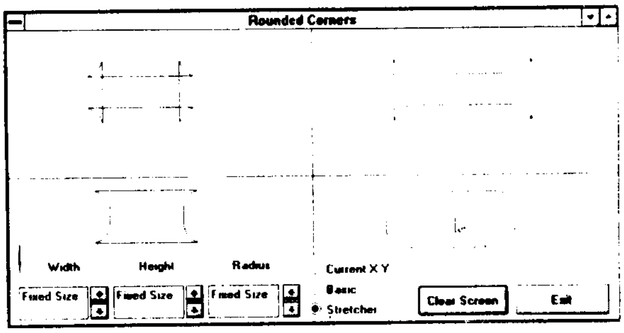

Rounded Corners
Width
Height
Radius
Current X Y
Fixed Size
Fixed Size
Fixed Size
Basic
● Stretcher
Clear Screen
Exit

图 11.1　Rounded Corners 程序的屏幕显示

三种方法的绘图过程如下:

1. CurrentXY:在 MouseDown 事件发生时,从当前的 X,Y 坐标开始画一条线;接着用 Pset 语句更新 CurrentX 和 CurrentY 的值,并在直线后接着画一个 1/4 圆弧,继续这一过程,按顺时针方向画出所有的四条线及圆弧,构成一个完整的圆角矩形,CurrentXY 过程适用于通过 MouseDown 事件放置圆角矩形且该圆角矩形不需被缩放或改变大小。
2. Basic:MouseDown 事件发生时,从 X,Y 点画一条线,接着画一 1/4 圆弧并按顺时针方向继续,直至绘出整个圆角矩形。Basic 过程用于绘制通过 MouseDown 事件改变大小的圆角矩形。
3. Stretcher:通过 MouseMove 的 X1,Y1,X2,Y2 的坐标值画出圆角矩形的各条边及圆弧,你可以通过鼠标放大或缩小圆角矩形,Stretcher 过程适用于绘制通过拖拉鼠标改变大小的圆角矩形。

11.1　如何使用本程序

现在已经准备好使用 Rounded Corners 程序,在本节中,将学到建立及修改圆角矩形所使用的窗体、控制对象的属性。下面是用来控制圆角矩形大小及形状的三个滚动条:

Width Scroll bar:用来改变圆角矩形的水平线的长度。

Height Scroll bar:用来改变圆角矩形的竖直线的长度。

Radius Scroll bar:用来改变圆角矩形的 1/4 圆弧的大小,该值如果置为 0 将引发错误。

CurrentXY 和 Basic 两个例程绘出的圆角矩形形状完全相同,但这两者的绘制过程略有

不同，具体见下文。

表 11.1 给出 Rounded Corners 程序窗体及其控制对象的属性。

表 11.1　Rounded Corners 程序窗体及其控制对象的属性

Object: Form
Object Name: Form1
Caption ="Rounded Corners"　　ScaleMode =3 'Pixel
Height =4920　　Top =1125
Left =60　　Width =9615

Object: Command Button
Object Name: Exit
Caption ="Exit"　　Top =4005
Height =465　　Width =1455
Left =7920

Object: Option Button
Object Name: Option1
Caption ="Stretcher"　　Left =4680
Height =330　　Top =4185
Index =2　　Width =1500

Object: Option Button
Object Name: Option1
Caption ="Basic"　　Left =4680
Height =330　　Top =3870
Index =1　　Width =1500

Object: Option Button
Object Name: Option1
Caption ="Current X－Y"　　Top =3555
Height =330　　Value =－1 'True
Index =0　　Width =1500
Left =4680

Object: Command Button
Object Name: Cls
Caption ="Clear Screen"　　Left =6390
Default =－1 'True　　Top =4005
Height =465　　Width =1410

Object: Textbox
Object Name: Text1
Height =510　　Width ="40"
Index =0　　Top =3960
Left =135　　Text =1095

Object: Textbox
Object Name: Text1
Height =510　　Text ="10"
Index =1　　Top =3960
Left =1620　　Width =1095

Object: Textbox
Object Name: Text1
Height =510　　Text ="26"
Index =2　　Top =3960
Left =3105　　Width =1095

```
Object: VScroll Bar
Object Name: VScroll1
     Height          =510                  SmallChange      =6
     Index           =2                    Top              =3960
     Left            =4260                 Value            =25
     Max             =1                    Width            =285
     Min             =600
Object: VScroll Bar
Object Name: VScroll1
     Height          =510                  SmallChange      =10
     Index           =1                    Top              =3960
     Left            =2745                 Value            =10
     Max             =0                    Width            =285
     Min             =600
Object: VScroll Bar
Object Name: VScroll1
     Height          =510                  SmallChange      =10
     Index           =0                    Top              =3960
     Left            =1260                 Value            =40
     Max             =0                    Width            =285
     Min             =600
Object: Label
Object Name: Label1
     Alignment       =2 'Center            Left             =3105
     Caption         ="Radius"             Top              =3555
     Height          =285                  Width            =1365
     Index           =2
Object: Label
Object Name: Label1
     Alignment       =2 'Center            Left             =1620
     Caption         ="Height"             Top              =3555
     Height          =285                  Width            =1365
     Index           =1
Object: Label
Object Name: Label1
     Alignment       =2 'Center            Left             =135
     Caption         ="Width"              Top              =3555
     Height          =285                  Width            =1410
     Index           =0
```

图 11.2 给出了窗体及其控制对象的屏幕显示图，该窗体及控制对象由清单 11.1 产生。

11.2 声明及初始值

窗体的声明部分（清单 11.1）给出八个整型数来存储每个圆角矩形的坐标值。

清单 11.1 初始变量名

```
Dim X1,Y1,X2,Y2,Wide,High,PI,Radius As Integer
```

Dim X1,Y1,X2,Y2：用于存储坐标轴的起点和终点的变量。

Wide,High:用于存储矩形的长度和宽度的变量。

PI,Radius As Integer:用于存储 PI(3.14159265)和圆的半径的变量。

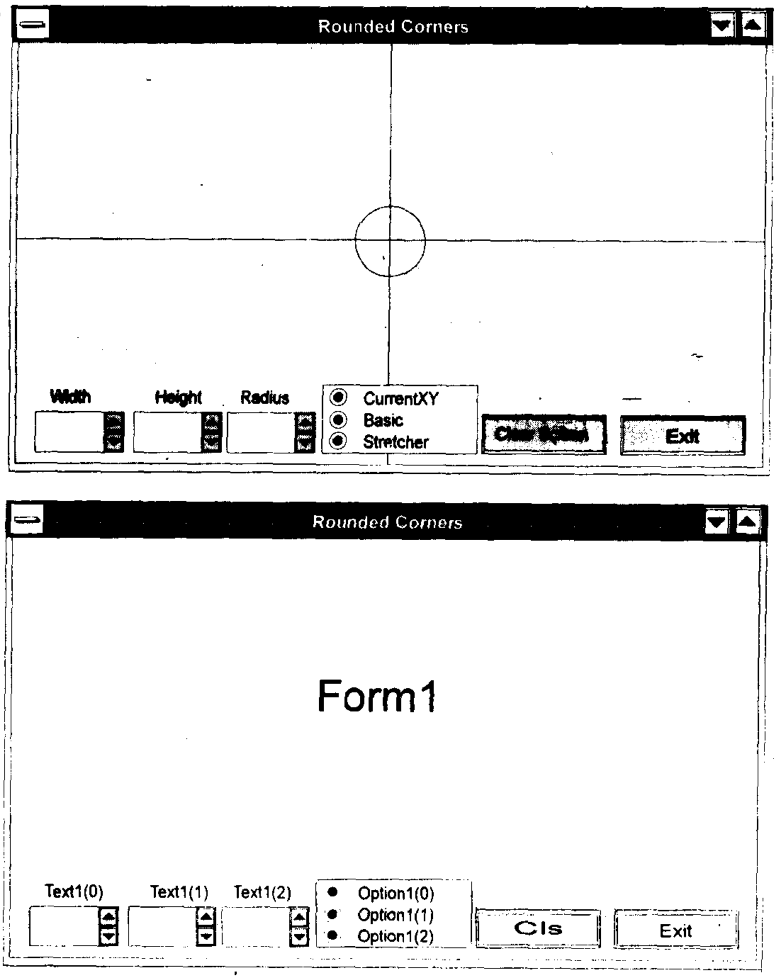

Rounded Corners
Width
Height
Radius
CurrentXY
Basic
Stretcher
Exit
Rounded Corners
Form1
Text1(0)
Text1(1)
Text1(2)
Option1(0)
Option1(1)
Option1(2)
Cls
Exit

图 11.2　窗体及其控制对象

Form_Paint 事件(见清单 11.2)画一条水平线及一条竖直线将窗体分为四个部分。(在 Stretch 过程中执行本段程序)。

清单 11.2　在窗体内画十字交叉线

```
Sub Form_Paint ()
    Line (0, ScaleHeight \ 2)-(ScaleWidth, ScaleHeight \ 2)
    Line (ScaleWidth \ 2, 0)-(ScaleWidth \ 2, ScaleHeight)
End Sub
```

Line:在窗体的正中心画两条十字交叉线。

Form_Resize 事件将窗体置于屏幕中心,见清单 11.3。

清单 11.3 将窗体置于屏幕中心

```
Sub Form_Resize ()
    Form1.Left = (Screen.Width - Width) \ 2
    Form1.Top = (Screen.Height - Height) \ 2
End Sub
```

Form1.Left：在水平方向上置窗体居中。

Form1.Top：在竖直方向上置窗体居中。

11.3 窗体的鼠标事件

选定选项按钮之后，Form_MouseDown 事件将调用不同的绘图过程，见清单 11.4。

清单 11.4 绘制一个圆角矩形

```
Sub Form_MouseDown (Button%, Shift%, X As Single, Y As Single)
  Select Case True
    Case Option1(0)
        DrawMode = 13: Cls
        CurrentXY Button, Shift, X, Y
    Case Option1(1)
        DrawMode = 13: Cls
        Basic Button, Shift, X, Y
    Case Option1(2)
        DrawMode = 6
        X1 = X: Y1 = Y
        X2 = X1: Y2 = Y1
        Stretcher Button, Shift, X, Y
    End Select
End Sub
```

Select Case True：根据三个 Option 控制对象的值，执行相应的 Case 程序。(当 Option 被选中时，为 True；否则，为 False)。

Case Option1(0)：选中了 CurrentXY 按钮。

DrawMode=Copy_Pen：画笔模式为 Copy_Pen(13)。

CLS：清屏。

CurrentXY：调用 CurrentXY 子程序(见清单 11.8)，该子程序通过 CurrentX，CurrentY 及 step 值和 Line 语句绘制圆角矩形。

Case Option1(1)：选中了 Basic 选项按钮。

DrawMode=Copy_Pen：置画笔模式为 Copy_Pen(13)。

CLS：清屏。

Basic：调用 Basic 子程序(见清单 11.6)，用精确的 X，Y 坐标来绘制圆角矩形。

Case Option1(2)：选中了 Stretcher 选项按钮。

DrawMode=6(Inverse)：置画笔模式为 Inverse 以便于你绘制移动物体。

X1，Y1，X2，Y2：存储鼠标当前的位置，坐标值将被用来作为移动圆角矩形的值。

Stretcher：调用 Stretcher 子程序来绘制圆角矩形，这个子程序与 Basic 过程一样，也是采用 Line 和 Circle 语句，它们的区别在于坐标值不同，Stretcher 取用 X1，X2，Y1，Y2，这样就产生了物体移动的效果。

当选项按钮为 Stretch 时，Form_MouseMove 事件（见清单 11.5）画出一圆角矩形。

清单 11.5　移动圆角物体

```
Sub Form_MouseMove (Button%, Shift%, X As Single, Y As Single)
If Button And Option1(2).Value Then
    Stretcher Button, Shift, X, Y
        X1 = X
        Y1 = Y
        X2 = X1 'remark out for skew effect
        Y2 = Y1 'remark out for skew effect
    Stretcher Button, Shift, X, Y
End If
End Sub
```

If Button And Option1(2)：MouseMove 事件仅在 Stretcher 过程（见清单 11.10）中被调用，故只有在鼠标被按下并且选项按钮被选中为 Stretcher 时，才执行下面的代码。

Stretcher Button,Shift,X,Y：画出圆角矩形物体。

X1,Y1,X2,Y2：置一条线的起点和终点为当前鼠标的坐标位置，绘制模式为 Inverse，这样第一次画出的图形与底色颜色相反，如果再在其上以底色画出同样的图形，就会形成移动的效果。

可以将 X2 和 Y2 的语句行以'号注释掉，这样就可以看到（skewing）的效果，撕拉是一种仅将物体在水平轴或竖直轴移动的方法。在本例中，撕拉改变了物体的形状，看起来就像是把物体在一个方向撕裂一样。

StretcherButton, Shift, X, Y：重新绘制圆角矩形对象，以便擦除上一次绘制的图形。

11.4　通过 Basic 过程画一个圆角矩形

清单 11.6 给出 Basic 过程的程序清单，由于 Line 语句在画线时，最后的一个点不被画上，故在整个图形中将留下空隙，所以在 Basic 过程中，使用 PSet 语句来填补这一空隙。但 Pset 象素点是方形的，最多有 4 个象素宽，然后把它以中心象素点为圆心变成圆形，因为排列均匀的象素点会在直线与圆弧的交接处产生不和谐的效果。在圆弧形状的绘制中，是从圆心点计算出来的。

在 Stretcher 过程中，使用的不是 Pset 语句而是变量 P1 和 P2。另一个可能的问题在于当设定的宽度值过大时，由于所有的线在两侧都与圆弧相连，就使得看起来像是有空隙。

清单 11.6　Basic 子程序

```
Sub Basic (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Const PI = 3.14159265
    Wide = Val(Text1(0).Text)
    high = Val(Text1(1).Text)
    Radius = Val(Text1(2).Text)
X = X - (Val(Text1(0).Text) \ 2) 'offsets from mouse pointer
```

```
Y = Y - Val(Text1(2).Text)
'''''''''''''''''''''

Line (X, Y)-(X + Wide, Y): PSet (X + Wide, Y)
Circle (X + Wide, Y + Radius), Radius, , 0, PI / 2
'''''''''''''''''''''

Line (X + Wide + Radius, Y + Radius)-(X + Wide + Radius, Y + Radius - high)
PSet (X + Wide + Radius, Y + Radius + high)
Circle (X + Wide, Y + high + Radius), Radius, , 3 * PI / 2, 0
'''''''''''''''''''''

Line (X + Wide, Y + Radius + high + Radius)-(X, Y + Radius + high + Radius)
PSet (X, Y + Radius + high + Radius)
Circle (X, Y + Radius + high), Radius, , PI, 3 * PI / 2
'''''''''''''''''''''

Line (X - Radius, Y + Radius + high)-(X - Radius, Y + Radius)
PSet (X - Radius, Y + Radius)
Circle (X, Y + Radius), Radius, , PI / 2, PI
End Sub
```

Const PI:常数值 PI=3.14157265。

Wide,High:矩形的宽与高。

Radius:圆角的半径。

下面给出清单 11.6 是怎样通过 Line 语句和 Circle 语句画成一个圆角矩形的。画圆角矩形的过程是顺时针进行的。

1. Line:起始的 X,Y 坐标值,从左到右画一条水平线。
2. Pset:在 Visual Basic 中,Line 语句中最后一个点不被画出,故使用 Pset 填充上该点。
3. Circle:(右上角)圆弧从时针的 12 点位置起始,终止于 3 点位置。
4. Line:从圆弧的 3 点位置起始,从上至下竖直地画一条线。
5. Pset:填充 Line 语句的最后一个点。
6. Circle:(右下角)圆弧从时针的 3 点位置起始,终止于 6 点位置。
7. Line:从圆弧的 6 点位置开始,从右到左水平地画一条线。
8. Pset:填充 Line 语句的最后一个点。
9. Circle:(左下角)圆弧从时针的 6 点位置开始,终止于 9 点位置。
10. Line:从圆弧的 9 点位置开始,从下至上竖直地画一条线。
11. Pset:填充 Line 语句的最后一个点。
12. Circle:(左上角)圆弧从时钟的 9 点位置开始,终止于 12 点位置。

清单 11.7 给出清除窗体内图形的程序。

清单 11.7　清除窗体内的图形

```
Sub Cls_Click ()
    Cls
    Form_Paint
End Sub
```

CLS:清屏。

Form_Paint：重画窗体及分割窗体的十字交叉线。

11.5　使用 Step 语句画圆角

清单 11.8 给出 CurrentXY 过程的清单，该程序使用 Step 语句连接直线与圆弧，使用 Pset 语句填充直线与圆弧连接处的空隙，Pset 语句还改变 CurrentX 和 CurrentY 坐标值以便和屏幕上画出的最后一点的位置相同。

清单 11.8　CurrentXY 子程序

```
Sub CurrentXY (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Const PI = 3.14159265
    Wide = Val(Text1(0).Text)
    high = Val(Text1(1).Text)
    Radius = Val(Text1(2).Text)
    CurrentX = X: CurrentY = Y
''''''''''''''''''''''''''

Line -(CurrentX, CurrentY - high)
Circle Step(-Radius, 0), Radius, , 0, PI / 2
''''''''''''''''''''''''''''''

PSet Step(0, -Radius)
Line -(CurrentX - Wide, Y - high - Radius)
Circle Step(0, Radius), Radius, , PI / 2, PI
'''''''''''''''''''''

PSet Step(-Radius, 0)
Line -(CurrentX, CurrentY + high)
Circle Step(Radius, 0), Radius, , PI, 3 * PI / 2
'''''''''''''''''''''

PSet Step(0, Radius)
Line -(CurrentX + Wide, CurrentY)
Circle Step(0, -Radius), Radius, , 3 * PI / 2, 0
End Sub
```

Const PI：常数值 PI=3.14159265。

Wide,High：矩形的宽与高。

Rodius：改变圆角的半径。

CurrentX=X:CurrentY=Y：新的 X,Y 坐标值。

下面是以顺时针方向画出圆角矩形的过程，本过程使用了窗体的 CurrentX 和 CurrentY 值，画的顺序为顺时针。

Line—：符号"—"表示从 CurrentX 和 CurrentY 点开始画线至右上角的圆弧。

Circle Step：Step 关键字表示从 CurrentX 和 CurrentY 位置偏移一定的距离至下一个位置。偏移量由 Step 后面的参数中给出，程序中以顺时针顺序依次画出四条边及圆弧。

PSet Step(0,—Radius)：Step 关键字表示从 CurrentX 和 CurrentY 位置偏移一定的距离至下一个位置，偏移量由 Step 后面的参数中给出，最后一个被画的物体为一个圆，故 CurrentX 和 CurrentY 为圆心的坐标。

与清单 11.8 中所用 CrrentX,CurrentY 的方法不同的另一种方法是通过 Line,Circle 和

Pset 语句中的 Step 关键字实现绘图过程，如下面给出的代码：

```
CurrentX=X:CurrentY=Y
'''''''''''''''''''''''''''''
Line -Step(0,-high)
Circle Step(-Radius,0), Radius,,0,PI/2
'''''''''''''''''''''''''''''
PSet Step(0,-Radius)
Line -Step(-wide,0)
Circle Step(0,Radius), Radius,,PI/2,PI
'''''''''''''''''''''''''''''
PSet Step(0,-Radius)
Line -Step(0,high)
Circle Step(0,Radius), Radius,,PI,3 * PI/2
'''''''''''''''''''''''''''''
PSet Step(0,-Radius)
Line -Step(wide,0)
Circle Step(0,Radius), Radius,,3 * PI/2,0
```

Step 关键字的起始值为当前的绘图位置，由 CurrentX 和 CurrentY 值给出；每当一条线，一段圆弧及一个点画下以后，CurrentX 和 CurrentY 的值就被改变，使用 Line 语句时，CurrentX，CurrentY 的值为线上最后一点的坐标值；使用 Circle 语句时，CurrentX 和 CurrentY 的值为圆心点的坐标，使用 PSet 语句时，CurrentX 和 CurrentY 的值为所画点的坐标。

11.6 选择一个绘制选项

当选中 Basic 或 CurrentXY 按钮时，窗体中的三个文本框内将显示出圆角矩形的宽、高及半径三个值；如果选中按钮 Stretcher，三个文本框将显示"Fixed"，表示圆角矩形的宽、高及半径的值将不会改变，见清单 11.9。

清单 11.9 选中某一个选项按钮

```
Sub Option1_Click (Index As Integer)
    If Index = 2 Then
        Cls: Form_Paint
        Text1(0).Text = "Fixed Size"
        Text1(1).Text = "Fixed Size"
        Text1(2).Text = "Fixed Size"
    Else
        Text1(0).Text = Vscroll1(0).Value
        Text1(1).Text = Vscroll1(1).Value
        Text1(2).Text = Vscroll1(2).Value
    End If
End Sub
```

If Index=2 Then：宽、高及半径值将不会被改变。

Else：宽高及半径值可被改变。

11.7 用 Stretcher 过程绘制圆角矩形

清单 11.10 给出 Stretcher 过程的程序，在该子程序中，使用 P1，P2 变量来填充 Line 语

句最后一个点的空隙。因为直线和圆弧的端点通常是圆的，如果选用的宽度值太大，将会使直线与圆弧的连接点处会看起来像有一段空隙。

清单 11.10　Stretcher 过程

```
Sub Stretcher (Button As Integer, Shift As Integer, X As Single, Y As Single)
Const PI = 3.14159265
Wide = X1 - (Form1.ScaleWidth \ 2)
high = Y1 - (Form1.ScaleHeight \ 2)
Radius = X2 - (X2 - 20)
If DrawWidth = 1 Then
    p1 = DrawWidth: p2 = DrawWidth 'offset pixel
Else
    p1 = DrawWidth: p2 = -DrawWidth 'offset pixel
End If
''''''''''''''''''''''
Line (X1 + p1, Y1)-(X2 + Wide + p2, Y2)
Circle (X1 + Wide, Y1 + Radius), Radius, , 0, PI / 2
''''''''''''''''''''''
Line (X1 + Wide + Radius, Y1 + Radius + p1)-[IC:CCC]
    (X2 + Wide + Radius, Y2 + Radius + high + p2)
Circle (X1 + Wide, Y1 + high + Radius), Radius, , 3 * PI / 2, 0
''''''''''''''''''''''
Line (X1 + Wide - p1, Y1 + Radius + high + Radius)-[IC:CCC]
    (X2 - p2, Y2 + Radius + high + Radius)
Circle (X1, Y1 + Radius + high), Radius, , PI, 3 * PI / 2
''''''''''''''''''''''
Line (X1 - Radius, Y1 + Radius + high - p1)-(X2 - Radius, Y2 + Radius - p2)
Circle (X1, Y1 + Radius), Radius, , PI / 2, PI
End Sub
```

Const PI：常数值 PI=3.14159265。

Wide.High：矩形的宽与高，在本例中，可以通过鼠标改变宽与高的值，原点位于窗体的正中心。

在窗体中心单击鼠标，则画出一个完整的圆；如果在中心的左侧单击鼠标，取用的宽度值为负；右侧单击鼠标，取用的宽度值为正；如果在窗体的上半部单击鼠标，取负的高度值，下半部单击鼠标，取高度值为正。

Radius：圆角的半径。

If DrawWidth=1 Then：当 DrawWidth 为 1 时，P1，P2 均为正；在其他的情况下，P1 值为正而 P2 值负。

Step 关键字的使用同 Basic 过程，在 Stretcher 过程中不使用 Pset 语句来填充 Line 语句造成空隙，而是采用 P1，P2 变量的方法。

11.8　使用滚动条控制对象

滚动条的 Value 属性给出圆角矩形的宽、高及圆弧半径。

清单 11.11 改变物体的大小

```
Sub VScroll1_Change (Index As Integer)
If Text1(0).Text = "Fixed Size" Then Exit Sub
    Select Case Index
        Case 0
            Text1(0).Text = Vscroll1(0).Value
        Case 1
            Text1(1).Text = Vscroll1(1).Value
        Case 2
            Text1(2).Text = Vscroll1(2).Value
    End Select
End Sub
```

Case 0：改变圆角矩形的宽。宽的最大值为 600，最小值为 0。当按向上的箭头时，值增加，按向下的箭头时，值减小。

Case 1：改变圆角矩形的高度，最大的高度值为 600，最小的值为 0，当单击向上的箭头时，高度值增大。

Case 2：改变圆角矩形四个圆弧的半径，最大值为 600，最小值为 1，当单击向上的箭头时，半径值增大。注意，半径值不能小于 1，否则将发生错误。

11.8 本章小结

在窗体内单击鼠标时，Basic，CurrentXY 和 Stretcher 过程就画出一个圆角矩形，如果想在圆角矩形的角上增加八个小的图片框控制对象，只需在 MouseMove 事件中把处理圆角的部分改一下即可，你可能还需要增加八个新变量以存储坐标值。

你也可以试着改变一下绘制圆角矩形的顺序，但是通过 Circle 语句画的圆只能以顺时针顺序画出。

因为 Visual Basic 中可使用 GDI32 API 函数，使用 API ArcTo 和 PolyDraw 函数可以缩短本程序的代码行，详细的 GDI32 API 函数可参见附录 B“Windows NT，Windows95 及未来的 Windows 版本”。

连接线与线、圆与圆、线与圆有很多方法，通过 CurrentX，CurrentY 坐标，使用图形方法可以通过很多途径实现它，可以查看第 1 章的总结，在那里给出了 Line，Circle 和 PSet 语句关于移动、连接等的详细内容。

第 12 章　文本对齐程序

本章主要介绍包括了文本的对齐方式、属性及绘制。有许多插图者和桌面出版者不仅使用文本来增强图形效果，而且还把它作为图形说明使用。

TxtAlign，APItext 和 DrawText 程序使用 Visual Basic 的 Print 语句和少量的 API 字体函数来对齐及格式化文本字符。如果是编写一个完整的字处理程序或是页面排版程序，就需要用高级 API 字体函数来写更复杂的过程。为了达到真实的打印预览(WYSIWYG)，就必须彻底了解高级字体技术。

Windows 使用逻辑度量单位在监视器上显示文本。此逻辑单位要比实际打印的字体大 40%左右，这是为了补偿较低的显示器分辨率而要求的。可以用 API 函数 GetDeviceCaps 来得到诸如显示器和打印机等设备的物理尺寸和逻辑尺寸。

使用 API 函数 OutLineTextMetrics 和 GetTextMetrics 可以得到字体的图形信息，这些信息包括正确的字母间距、行间距(Leading)、字符位置及纵横比等。

为了把字符点阵大小(每英吋 72 点)转换到显示器尺寸，可用 API 函数 GetDeviceCaps 得到显示器的 LOGPIXELSY(垂直方向上每英吋象素个数)，然后使用如下公式：

```
API=GetDeviceCaps(ArtWork.hDC,90)
ifHEIGHT=-1 * (API * point size)/72)
```

1. 值 90 表示 LOGPIXELSY 索引。
2. ifHEIGHT 变量是 LOGFONT 结构的成员。LOGFONT 结构保存将绘于屏幕上的所有字体的属性。
3. 负值(-1)告知 GUI 使用字符的高度而非包围每个字符的不可见边界框(单元)的高度。
4. 变量(point size)可以是输入文本框的一个数字。例如：

```
ifHeight=-1 * (API * Text1.Text)/72
```

12.1　行间距(Leading)

行间距(Leading)是指文本中每两行之间的间距大小。在字处理程序中可用 1 行、1.5 行、2 行等值改变行间距值。通过计算每段文本前、后零碎间隙还可以得到附加的印刷效果。

要在多行文本中使用默认的行间距值，可用下面公式计算 Leading：

```
Leading=tmHeight+tmExternalLeading
```

然后将 Leading 乘以此行的行号即可得到其与文本首行的间距。例如(3 * Leading)可准确获知第 3 行文本的 Y 向文本位置。

下面是 TextOut 函数中 Leading 使用的示例：

```
LineCounter =LC
Leading=tmHeight+tmExternalLeading
```

```
Storedtext=Lable1.Caption
TxtLength=Len(Label1.Caption)
API=TextOut(form1.hDC,X,LineCounter * Leading,Storedtext,TxtLength)
```

行数可用 API 函数 SendMessage 从一些 Visual Basic 控制对象中获取 EM_LINECOUNT 或由自己编码处理每行文本的输入与删除来得到。Artisan 程序使用了后一种方法，其方式是检测回车键与退格键。但这种方式下不能使用块删除功能。

12.2 字间距(Width)

某些文本字符串打印输出时的长度与显示器上的显示长度不一致。在这种情况下，使用 API 的 GetCharABCWidth 函数调整每个字符的“ABC”尺寸。所谓“ABC”尺寸，简单地说就是每个字符占用的总宽度。请记住，每个字符都位于一个不可见的限制框中，“A”是指框左边的间隙，“B”指字符的宽度，“C”指右边的间隙。若设置“A”或“C”为负值，字符的边隙就减小，这使得字间距变小，字符排列更为紧凑。

可认为 tmInternalLeading 值是字符边界框(或单元)上下底之间的间隙。内隙(internal leading)是指字符显示高度与字符边界框(或单元)高度之差。显示高度是字符显示在显示器上的高度。

文本中行之间的空隙(White space or leading)称为 tmExternalLeading，外隙(External leading)不是字符边界框单元的一部分。在不能直接指定某种字体的字符单元时，可以用内隙(internal leading)加上外隙(External leading)来计算这种字体的字间距(leading)。

表 12.1 TxtAlign 窗体及其控制对象属性

```
Object: Form
Object Name: Form1
    Caption      ="Alignment"       Height       =2730
    Left         =645               ScaleMode    =3 'Pixel
    Top          =1215              Width        =4920

Object: Frame
Object Name: Frame1
    Caption      ="Alignment"       Height       =1320
    Left         =3330              Top          =90
    Width        =1410

Object: Option Button
Object Name: Option1
    Caption      ="Right"           Height       =240
    Index        =2                 Left         =90
    Top          =990               Width        =1230

Object: Option Button
Object Name: Option1
    Caption      ="Center"          Height       =240
    Index        =1                 Left         =90
    Top          =675               Value        =-1 'True
    Width        =1230

Object: Option Button
Object Name: Option1
```

Caption ="Left"　Height =240
Index =2　Left =90
Top =315　Width =1230

Object: Picture Box
Object Name: Picture1
Height =2130　Left =120
MousePointer ='Cross　ScaleMode =3 'Pixel
Top =120　Width =3120

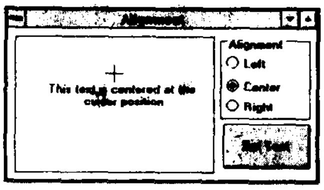
Object: Label
Object Name: Label1
AutoSize =-1 'True　BorderStyle = 'Fixed Single
Caption ="Type something"　Height =225
Left =810　Top =540
Width =1500

Object: Command Button
Object Name: SetTextBtn
Caption ="Set Text"　Height =690
Left =3375　Top =1485
Width =1365

12.3 TxtAlign 程序

TxtAlign 程序使用 Visual Basic 的 Print 语句，用人工调整计算的方式在图片控制对象 Picture1 中放置文本。

图 12.1 为 TxtAlign 程序的屏幕显示效果。



This text is centered at the cursor position

图 12.1　显示 TxtAlign 程序屏幕

清单 12.1 表明了怎样设置 TxtAlign 程序的默认变量。

清单 12.1　startup 中的变量名

```
Declaration
Dim TextX As Single
Dim TextY As Integer
```

Dim TextX, TextY：保存图片框控制对象 Picture1 中的 CurrentX 和 CurrentY 坐标值。这两个值给出第一行文本起始打印位置的坐标。

清单 12.2 是 Form_Resize 事件的程序，它使窗体定位于屏幕的中心，并指定输入文字所在的位置。

清单 12.2 窗体居中与文本光标(caret)定位

```
Sub Form_Resize ()
    Left=(Screen. Width - Width)\ 2
    Top=(Screen. Height - Height) \2

    TextX=Label1. Left
    Picture1. CurrentY=Label1. Top+1
    Label1. FontSize=Picture1. FontSize
    Picture1. SetFocus
End Sub
```

Left. Top:用于将窗体显示在屏幕正中。

TextX=Label1. Left:录入文本时,文本被置入一个标题控制对象中。单击对齐选项按钮或设置文本按钮时,文本被传入图片框 Picturel 中。TxtAlign 程序使用 Visual Basic Print 语句绘制文本。TextX 变量为 Print 语句提供 CurrentX 坐标。要使标题控制对象中的文本在图片框 Picturel 中得到相同的坐标位置,只须使 Text1 变量值与标题控制对象的 Left 属性相等即可。

Picturel. CurrentY=Label1. Top+1:基本上与 TextX 相同,只是标题控制对象的边界宽度增加一个象素单位。

Label1. FontSize=Picture1. FontSize:使标题控制对象的字体尺寸与 Picture1 的字体尺寸匹配。

清单 12.3 中的 Option1_Click 事件强迫设置文本(Set Text)按钮执行文本的对齐与绘制。

清单 12.3 用选项按钮对齐文本

```
Sub Option1_Click(Index As Integer)
    SetTextBtn_Click
End Sub
```

SetTextBtn_Click:驱动 TextBtn_Click(清单 12.6)事件以执行文本绘制。

清单 12.4 中 Picture1_KeyPress 事件用以显示标题控制对象中输入的每个字符。

清单 12.4 在标题控制对象中输入字符

```
Sub Picture1_KeyPress(KeyAscii As Integer)
    If Label1. Caption="Type something" Then Label1. Caption=""
    Label1. Caption=Label1. Caption+Chr(KeyAscii)
End Sub
```

If Labl1:提示用户开始输入。

Label1. Caption:键入的字符被显示于标题控制对象中。

清单 12.5 中 Picture1_MouseDown 事件清除图片框中的文本并把标题控制对象移到鼠

标指针指向的坐标处。

清单 12.5　设定文本输出时的起点位置

```
Sub Picture1_MouseDown(Button%,Shift%,X!,Y!)
    Picture1.AutoRedraw=True
    Picture1.Cls
    Picture1.AutoRedraw=False
    TextX=X:Picture1.CurrentY=Y
    Label1.Left=X:Label1.Top=Y-1
    Label1.Height=15:Label1.Width=2:Label1.Visible=True
End Sub
```

Picture1. AutoRedraw：绘制多行文本时，应设 AutoReDraw 属性为 True，并用 Cls（清屏方法）擦除旧的文本图象。

TextX，CurrentX：文本从鼠标当前位置开始显示。

Label 1. Left（Top）：将标题控制对象移到文本起始位置（当前鼠标指针所指的位置）。

Label 1. Height，Label 1. Width：使标题外观像一个文本光标（a text cursor）。

Label 1. Visible：显示标题控制对象。

图 12.2 显示了 TxtAlign 窗体及其控制对象。

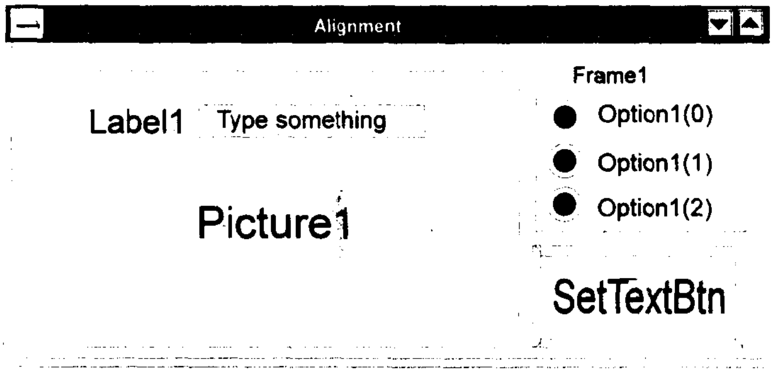


图 12.2　TxtAlign 窗体及其控制对象

12.4　在 Visual Basic 中手工放置文本

清单 12.6 中的 SetTextBtn_Click 事件将键入标题控制对象中的每行文本提取出来。TextWidth 方法计算出每行文本的长度，然后加上一个补偿偏移量以使文本左对齐、居中或右对齐，而后将提取出的这行文本打印在 Picture1 图片框中。最后，计算下一个 CurrentX 和 CurrentY 坐标值以确定下一行文本的位置。此过程重复执行以处理标题控制对象中现存的每行文本。

清单 12.6　从标题控制对象中提取文本

```
Sub SetTextBtn_Click
    If Label1.Caption="" Then Exit Sub
    Dim LineCount As Integer
    LabelWidth=Label1.Width
    Label1.Visible=False
For R=1 To 100 'LineCount Maximum
    Labeltext$=Label1.Caption+Chr(13)
    CR=InStr(Labeltext$,Chr(13))
    TopLine=Left(Labeltext$,CR-1)
    Select Case True
        Case Option1(0).Value
            Picture1.CurrentX=TextX
        Case Option1(1).Value
            Picture1.CurrentX=TextX-(TextWidth(TopLine)\2)
        Case Option1(2).Value
            Picture1.CurrentX=TextX-(TextWidth(TopLine))
        End Select
Picture1.AutoRedraw=True
Picture1.Print TopLine
Picture1.AutoRedraw=False
    txtLength=Len(Labeltext$)
    If txtLength=LineCount Then Label1.Caption="":
    Picture1.SetFocus:Exit Sub
    LineCount=LineCount+1
    Scroll=Mid(Labeltext$,CR+1,txtLength)
    Label1.Caption=Scroll
Next R
End Sub
```

If Label1.Caption：未键入字符，故退出此事件。

Dim LineCount As Integer：For/Next 语句块的循环次数。可修改此值使其等于标题控制对象中现存文本的总行数。

Label1.Visible：在即将向 Picture1 中传送标题控制对象里的内容时将标题控制对象隐藏起来。

For R=1 To 100：对从 1 到 100 的每个文本行执行下面的代码段（本例中 R 最大值取 100）。

Labeltext$：表示标题控制对象中输入的所有文本。在文本末尾加一个回车（Chr(13)）是为了设置一个“搜寻”（Search for）标记以便使用 InStr 函数。

CR=InStr：返回回车符（Chr(13)）的位置。InStr 函数的语法如下：

InStr(the text we are searching in, the character we are Looking for)

TopLine=Left(Labeltext$,CR-1)：返回标题控制对象中最顶上的一行文本（减去 Chr(13)）。

Select Case True：根据三个选项控制对象的值（True 表示选项被选中；False 表示选项未被选中），转向 Case 表达式为真的语句。

Case Option1(0)：选中左对齐选项按钮。

Picture1.CurrentX：文本左对齐于屏幕位置 CurrentX。

Case Option1(1)：选中居中选项按钮。

Picture1.CurrentX：最长文本行上、下方的文本与此行文本居中对齐。

Case Option1(2):选中右对齐选项按钮。

Picturel. CurrentX:最长文本行上、下方的文本与此行文本右对齐。

Picturel. AutoRedraw:设为 True,此时重复执行 Print 语句时新输入的每行文本不会被擦除。

Picture1. Print:打印第一行文本。

Picture1. AutoRedraw:置 AutoRedraw 为 off,否则每行文字都会出现两个图象。

txtLength:表示标题文本的长度(字符数)。

If txtLength:行计数值达到终值时擦除标题(Caption),然后退出子程序。

Line Count:行计数值加 1。

Scroll:重绘第一行文本之后的每一行标题文本。

Label1. Caption:清除标题控制对象中已绘入图片框控制对象 Picturel 中的首行文本,然后载入下一行文本做为首行文本。

Next R:重复此过程至绘完所有文本。

12. 5　TxtAlign 程序总结

如果不想在程序中使用 API 的文本对齐函数,那么 TxtAlign 程序中提供的例程足以完成简单的文本排版任务。注意 Scroll 变量(清单 12. 6 中的 SetTextBtn_Click)使用 Mid $ 字符串函数模拟文本卷动的处理方法。每行文本绘制后即从标题控制对象中删除,标题控制对象的标题自动地把后面的文本上移一行。

SetTextBtn_Click 事件只是提取标题控制对象中的一行标题(Caption)并用 Visual Basic 的 Print 语句重绘入另一控制对象的一个例子。你可以将 Visual Basic 字符串函数 Instr $ 、Left $,LTrim,RTrim,Mid $ 和 Right $ 进行替换组合(alternative combination),然后加以试验。

12. 6　使用 APItext 程序

APItext 程序把 API 文本绘制功能和 Basic 语言的 Print 语句结合起来。APItext 程序使用了一个标准的 API 文本格式函数。对于专业字体操作,还须对高级字体技术有一个彻底的了解。

API 的 GlyphOutLine 函数是 API 的一个例子,用于连接把文本字符做为图形对象进行操作的高级例子。API 函数 GlyphOutLine 函数从每个文本字符的轮廓构造中提取信息。

下面这些步骤生成操作一种字体所需的数据结构:

1. 输入 POINTAPI (X,Y)
2. 输入 FIXED (fract,value)
3. 输入 mat2(FIXED)
4. 输入 POINTFX(X,Y)
5. 输入 GLYPHMETRICS
6. 输入 TTPOLYGONHEADER
7. 输入 TTPOLYCURVE

获取字符轮廓所需的附加分量(elements)是：

- 一个 B-Spline Bézier 曲线过程
- 多边形端点和多义线管理

若想深入了解 TrueType 字体的内部工作机制，则需要参考另外的文献。由于提取一个 TrueType 字符的结构需要大量例程，本章没有提供范例。FONT.BAS 文件列出大多数的文本类型结构(Type structures)。

APItext 程序的窗体和控制对象属性列于表 12.2 中。

表 12.2 APItext 窗体及其控制对象属性

Object: Form			
Object Name: TextEditor			
BorderStyle	=1 'Fixed Single	Caption	="Text Editor"
Height	=5955	Left	=225
ScaleMode	=3 'Pixel	Top	=1185
Width	=8625		
Object: Check Box			
Object Name: Check1			
Caption	="Rotation"	Height	=195
Index	=5	Left	=6780
Top	=4890	Width	=1230
Object: Check Box			
Object Name: Check1			
Caption	="Weight"	Height	=195
Index	=4	Left	=6780
Top	=4665	Width	=1230
Object: Check Box			
Object Name: Check1			
Caption	="Width"	Height	=195
Index	=3	Left	=5520
Top	=4890	Width	=1230
Object: Check Box			
Object Name: Check1			
Caption	="Underline"	Height	=195
Index	=2	Left	=5520
Top	=4665	Width	=1230
Object: Check Box			
Object Name: Check1			
Caption	="StrikeThru"	Height	=195
Index	=1	Left	=4215
Top	=4890	Width	=1230
Object: Check Box			
Object Name: Check1			
Caption	="Italic"	Height	=195
Index	=0	Left	=4215
Top	=4665	Width	=1230
Object: Command Button			

Object Name: FontLogBtn

```
Caption          ="Font Log"          Enabled      =0 'Flase
Height           =360                 Left         =5580
Top              =5160                Width        =1320
```

Object: Command Button
Object Name: APICRLF

```
Caption          ="API CR/LF"         Height       =360
Left             =2820                Top          =5160
Width            =1320
```

Object: Command Button
Object Name: TextMex

```
Caption          ="Text Metrics"      Enabled      =0 'Flase
Height           =360                 Left         =4200
Top              =5160                Width        =1320
```

Object: Picture Box
Object Name: Artwork

```
Height           =3435                Left         =60
MousePointer     =2 'Cross            ScaleMode    =3 'Pixel
Top              =60                  Width        =5535
```

Object: Textbox
Object Name: Text1

```
FontSize         =9.75                Height       =3435
Left             =5700                MultiLine    =-1 'True
ScrollBars       =3 'Both             Top          =60
Width            =2775
```

Object: Command Button
Object Name: APItext

```
Caption          ="API text"          Height       =360
Left             =1440                Top          =5160
Width            =1320
```

Object: Combo Box
Object Name: Style_Combo

```
Height           =300                 Left         =4260
Style            =2 'Dropdown List    Top          =3720
Width            =1905
```

Object: Frame
Object Name: Frame1

```
Caption          ="Alignment"         Height       =1485
Left             =2460                Top          =3600
Width            =1695
```

Object: Option Button
Object Name: Option1

```
Caption          ="Right"             Height       =375
Index            =2                   Left         =120
Top              =1020                Width        =1095
```

Object: Option Button
Object Name: Option1

```
Caption          ="Center"            Height       =375
Index            =1                   Left         =120
Top              =660                 Width        =1095
```

Object: Option Button
Object Name: Option1

```
Caption      ="Left"            Height        =330
Index        =0                 Left          =120
Top          =360               Value         =-1 'True
Width        =1095
```

Object: Command Button
Object Name: Option1 CloseBtn

```
Caption      ="Close"           Height        =360
Left         =7080              Top           =5160
Width        =1335
```

Object: Command Button
Object Name: BasicPrint

```
Caption      ="Basic print"     Height        =360
Left         =60                Top           =5160
Width        =1335
```

Object: VScroll Bar
Object Name: Vscroll1

```
Height       =1365              LargeChange   =10
Left         =8145              Max           =4
Min          =400               Top           =3690
Value        =12                Width         =300
```

Object: List Box
Object Name: FontList

```
Height       =1470              Left          =60
Sorted       =-1 'True          Top           =3600
Width        =2295
```

Object: Label
Object Name: SizeBox

```
Alignment    =1 'Right Justify  BorderStyle   =1 'Fixed Single
Height       =315               Left          =7080
Top          =3720              Width         =975
```

Object: Label
Object Name: ViewFont

```
Height       =480               Left          =4260
Top          =4095              Width         =3735
```

Object: Label
Object Name: Label3

```
Alignment    =1 'Right Justify  Caption       ="Size:"
Height       =255               Left          =6240
Top          =3690              Width         =50
```

12.7 APItext 程序中的声明与结构

APItext 程序利用标准 API 文本函数处理标准字体文本字符的放置与对齐。此程序使用 API 函数对齐、旋转、加下划线和实现其他几种文本格式化过程。API 函数 GetTextMetrics 和 CreateFontIndirect 在此程序中用于提供当前字体信息或向一种字体添加特殊属性。

清单 12.7 初始化 APItext 程序中使用的变量。

清单 12.7　初始化时的变量名

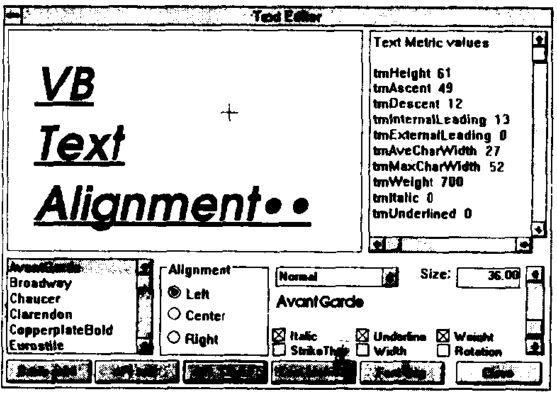
```
Declaration
Dim Align As Integer
Dim APIcr As FontLog
Dim TextX,TextY As Single
```

Dim Align：将文本对齐方式告知 API 函数 SetTextAlign。

Dim APIcr：使用 FontLog 的 Type 结构保存 API 字体属性。FontLog 结构可在 FONT.BAS 文件中找到。

Dim TextX,TextY：保存文本的起始位置，然后将此值传给 API 函数 TextOut。

图 12.3 为 APItext 程序屏幕显示的例子。


Text Editor
VB
Text
Alignment• •
AvantGarde
Broadway
Chaucer
Clarendon
CopperplateBold
Eurostile
Alignment
● Left
○ Center
○ Right
Normal
Size:
36.00
AvantGarde
☒ Italic
☒ Underline
☒ Weight
☐ StrikeThru
☐ Width
☐ Rotation
Close

```
Text Metric values

tmHeight 61
tmAscent 49
tmDescent 12
tmInternalLeading 13
tmExternalLeading 0
tmAveCharWidth 27
tmMaxCharWidth 52
tmWeight 700
tmItalic 0
tmUnderlined 0
```

图 12.3　APItext 程序的屏幕显示

12.8　默认的初始值

程序初始化时将系统安装的前十种屏幕字体载入一个列表框中。若系统中安装的字体不足十种，则必须修改初值使其等于系统实际安装的字体数。

清单 12.8　装载字体列表和赋默认初值

```
Sub Form_Load ()
    Style_Combo.AddItem "Normal"
    Style_Combo.AddItem "Bold"
    Style_Combo.AddItem "Italic"
    Style_Combo.AddItem "Bold-Italic"
    SizeBox.Caption = "12.00"
    ViewFont.FontSize = 12
For R = 0 To 10
    FontNames$ = Screen.Fonts(R)
    FontList.AddItem FontNames$
Next R
    FontList.ListIndex = 0
    Style_Combo.ListIndex = 0
End Sub
```

Style_Combo. AddItm：把标准、斜体和粗体属性名加入组合列表框中。

SizeBox. Caption＝"12. 00"：绘制文本时的默认字体大小。

ViewFont. FontSize＝12：ViewFont 标题控制对象显示当前选中字体的图象。

For R＝0 To 10：向列表框控制对象 FontList 中加入 10 种屏幕字体。

FontList. ListIndex＝0：高亮显示列表框控制对象中的第一种屏幕字体名。

Style_Combo. ListIndex＝0：选中 Style 组合框中的第一种字体属性（标准）。

图 12.4 示出了 APItext 窗体及其控制对象。

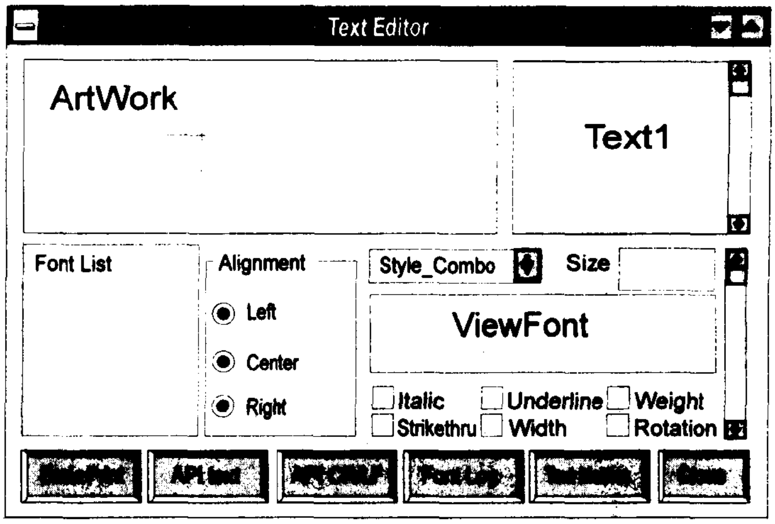

Text Editor
ArtWork
Text1
Font List
Alignment
Left
Center
Right
Style_Combo
Size
ViewFont
Italic
Underline
Weight
Strikethru
Width
Rotation

图 12.4　APItext 窗体及其控制对象

清单 12.9 中的 Form_Resize 事件将窗体显示于屏幕中央。

清单 12.9　把窗体显示于屏幕中央

```
Sub Form_Resize()
    TextEditor.Move(Screen.Width-Width)/2,[IC:CCC]
        (Screen.Height-Height)/2
End Sub
```

TextEditor. Move：将窗体精确定位于屏幕中央。

清单 12.10 使用 API 文本函数时用 Form_Unload 事件删除字体变化。

清单 12.10　退出例程前删除被改变设备环境属性

```
Sub Form_Unload (Cancel As Integer)
    R = DeleteObject(FontSelection%)
End Sub
```

DeleteObject：当一个 API 函数生成一种字体时，为字体所属控制对象分派一个初始字体对象。不使用此字体时须将此对象删除。

注意： 此例中使用了三种不同的字体函数。这些函数很容易搅乱 GUI。运行此例时不可同时运行另外的应用程序。如果删除 API 字体对象的命令与 Visual Basic 的内部字体管理发生冲突，则很容易使字体管理程序失控。

12.9 在 API Text 中添加回车换行符

当 API CR/LF按钮被按下时，APICRLF_Click 事件被触发。除了使用 API 文本函数外，此例程基本上与前述 TxtAlign 程序使用的文本打印例程相同。

首先，把捡取按钮中选中的字体属性载入 APIcr 变量，APIcr 指向 Type FontLog 结构以获取数据类型。然后执行下列步骤以绘制文本：

1. API 函数 CreateFontIndirect 创立字体，API 函数 SelectObject 设置设备环境属性。
2. API 函数 GetTextMetrics 获取每行文本所需的行距。
3. API 函数 SetTextAlign 计算文本的排版信息(justification)。
4. API 函数 SendMessage 计算文本框控制对象 Text1 中的文本行数。
5. API 函数 TextOut 绘制文本(见清单 12.11)。

清单 12.11　使用 API 函数 TextOut 绘制文本

```
Sub APICRLF_Click ()
    TextX = ArtWork.CurrentX
    TextY = ArtWork.CurrentY
If FontList.ListCount = 0 Then Exit Sub
'''''''''''''''''''''''

FontLogBtn.Enabled = True
APIcr.FontCharSet = Chr $ (DEFAULT_CHARSET)
APIcr.FontClipPrecision = Chr $ (OUT_DEFAULT_PRECIS)
APIcr.FontFaceName = FontList.Text + Chr $ (0)
API = GetDeviceCaps(ArtWork.hDC, 90)
APIcr.FontHeight = -1 * (API * Val(SizeBox.Caption)) \ 72
APIcr.FontItalic = Chr $ (Check1(0).Value)
APIcr.FontOutPrecision = Chr $ (OUT_DEFAULT_PRECIS)
APIcr.FontPitchFamily = Chr $ (DEFAULT_PITCH Or FF_DONTCARE)
APIcr.FontQuality = Chr $ (DEFAULT_QUALITY)
APIcr.FontStrikeThru = Chr $ (Check1(1).Value)
APIcr.FontUnderLine = Chr $ (Check1(2).Value)
If Check1(3).Value = 1 Then APIcr.FontWidth = 20 Else APIcr.FontWidth = 0
If Check1(4).Value = 1 Then APIcr.FontWeight = 700 Else APIcr.FontWeight = 0
If Check1(5).Value = 1 Then APIcr.FontRotate = 1800 Else APIcr.FontRotate = 0
''''''''''''''''''''''''''''''''

FontSelection% = CreateFontIndirect(APIcr)
NewFont% = SelectObject(ArtWork.hDC, FontSelection%)
    Dim tm As TEXTMETRIC
    API = GetTextMetrics(ArtWork.hDC, tm)
    Leading = tm.tmHeight + tm.tmExternalLeading
Select Case True
    Case Option1(0).Value
        R = SetTextAlign(ArtWork.hDC, TA_LEFT)
    Case Option1(1).Value
        R = SetTextAlign(ArtWork.hDC, TA_CENTER)
    Case Option1(2).Value
        R = SetTextAlign(ArtWork.hDC, TA_RIGHT)
End Select
'''''''''''''''''''''''''''''''''
Const WM_USER = 1024
Const EM_GETLINECOUNT = WM_USER + 10
TotalLines = SendMessage(Text1.hWnd, EM_GETLINECOUNT, 0, 0&)
For B = 1 To TotalLines
    textStr$ = Text1.Text + Chr(13)
    CR = InStr(textStr$, Chr(13))
    If B = CarriageCount Then
        TopLine$ = Left(textStr$, CR - 3)
    Else
        TopLine$ = Left(textStr$, CR - 1)
    End If
    R = TextOut(ArtWork.hDC, TextX, TextY + Cursor, (TopLine$), Len(TopLine$))
    Cursor = (B * Leading)
    TxtLength = Len(textStr$)
    LineCount = LineCount + 1
    Scroll = Mid(textStr$, CR - 2, TxtLength)
    Text1.Text = Scroll
Next B
    R = SelectObject(ArtWork.hDC, NewFont%)
    R = DeleteObject(FontSelection%)
End Sub
```

TextX,TextY：保存图片框的 CurrentX 和 CurrentY 并传给 API 函数 TextOut。文本的第一个字符定位于此。用 MouseDown 坐标可设置 TextX 和 TextY。

If FontList.ListCount= 0 Then：尽管不太可能，但或许系统中未安装字体。这种情况下退出此事件。

FontLogBtn.Enabled：指出当前字体属性。

下面是 FONT.BAS 文件中可找到的 API LogFont 结构的粗略轮廓(rough outline)：

APIcr.**FontCharSet**＝ANSI_Charset 或 Default_Chaset 或 Symbol_Charset 或 OEM_Charset。

APIcr.**FontClipPrecision**＝参见 FONT.BAS 清单。

APIcr:**FontFaceName**＝字体名，或者是缺省设备字体名。

APIcr.**FontHeight**＝以正值表示的字符单元高度，或用负值表示的 Glyph 高度。

APIcr.**FontItalic**＝0 表示不用斜体，其他任何值表示斜体。

APIcr.**FontOutPrecision**＝参见 FONT.BAS 清单。

APIcr.**FontPitchFamily**＝default_Pitch 或 fixed_Pitch 或 variable_Pitch 族：FF_Decorative，FF_Modern，FF_Roman，FF_Script，FF_Swiss 或 FF_Dontcare.

APIcr.**FontQuality**＝Default_Quality 或 Draft_Quality 或 Proof_Quality。

APIcr.**FontRotate**＝字体的旋转。

APIcr.**FontStrikeThru**＝0 表示不加删除线，其他任何值表示加删除线。

APIcr.**FontUnderLine**＝0 表示不加下划线，其他任何值表示加下划线。

APIcr. **FontWidth**＝以正值表示的字符单元宽度，或以负值表示的 Glyph 宽度。

APIcr. **FontWeight**＝从 100 到 900 的值。100 表示浅色，400 表示正常，700 表示粗体。

此例程中 API 函数 GetDeviceCaps(ArtWork. hDC，90)返回屏幕的 LogPixelsY 值。此值等于屏幕纵向每英寸逻辑象素数。使用此值可以按照所见即所得(WYSWYG)要求的点阵尺寸显示字体。在印刷测量单元中，每英寸有 72 点。

APIcr. FontHeight 等于下面的公式：

```
-1 * (API * Val(SizeBox. Text))\72
```

此公式给出准确的屏幕字体尺寸。

If Check1(index). Value＝1：粗体、斜体、宽体、加下划线、加删除线或旋转字体。用清单 12.11 中的例子可以改变第一个值(20)以增添或减少字体的属性。用最后一个值可以关闭或将字体设为标准属性。例如：

```
If Check1(3). Value=1 Then APIcr. FontWidth=20 Else APIcr. FontWidth=0
```

FontSelection%，NewFont：创建字体并设置设备环境属性。

API＝GetTextMetrics(ArtWork. hDC，tm)：获取 Artwork 图片框的字体属性。此时可获取当前字体的高度和行距值(leading)。一个字体的字符被置于其边界框单元的正中，边界框有左、上、右和下四条边。放置这四条边的同时就定义了字符周围的空白空间。行距(leading)不是边界框单元的一部分而是每行单元之间的距离。

Leading＝tm. tmHeight＋tm. tmExternalLeading：把 TextMetrics 提供的内隙(external leading)值加上字符高度得到准确的行距并把此行距提供给每一新行文本。

Select Case True：选择当前高亮显示的对齐选项按钮。

R＝SetTextAlign(ArtWork. hDC，TA_LEFT)：设置文本的对齐方式。

TotalLines＝SendMessage：获取文本框 Text1 中的总文本行数。

For B＝1 To TotalLines：例程的其他部分与本章开始的 TxtAlign 程序中的 SetTextBtn_Click 例程相似，只是 Artwork. Print 语句由 API 函数 TextOut 替代。

R＝TextOut：众多 API 文本函数之一，此函数在窗口中绘制文本。在此例中，文本被绘入名为 Artwork 的图片框中，第一行文本绘在位置 TextX(X 坐标)和 TextY＋Cursor(Y 坐标)处。

API 函数 TextOut 函数绘制文本的方式类似于 Print 语句。其格式如下：

```
TextOut(1,2,3,4,5)
```

1. 文本输向的控制对象(HDC)。
2. 文本起始点 X 坐标值。
3. 文本起始点 Y 坐标值。
4. 将要绘制的文本。
5. 文本字串的长度。

API 函数 TextOut 输出包含回车符和换行(Line Feeds)符的所有字符。

SelectObject：替代先前的对象类型(type)。

12.10　向 TextOut 函数添加属性

在清单 12.12 中，标有 API Text 的按钮基本上与清单 12.11 中 APICRLF_Click 事件相同，只是不删除文本中嵌入的回车符和换行符。要注意使用 API Text 函数时回车符(Chr(13)和换行符(Char(10))将被输出。在某些字体中它们的输出是方框或圆点符号。

清单 12.12　绘制嵌入 CR/LF 的文本

```
Sub APItext_Click ()
    TextX = ArtWork.CurrentX
    TextY = ArtWork.CurrentY
If FontList.ListCount = 0 Then Exit Sub
'''''''''''''''''''''''

TextMex.Enabled = True
APIcr.FontCharSet = Chr $ (DEFAULT_CHARSET)
APIcr.FontClipPrecision = Chr $ (OUT_DEFAULT_PRECIS)
APIcr.FontFaceName = FontList.Text + Chr $ (0)
API = GetDeviceCaps(ArtWork.hDC, 90)
APIcr.FontHeight = -1 * (API * Val(SizeBox.Caption)) \ 72
APIcr.FontItalic = Chr $ (Check1(0).Value)
APIcr.FontOutPrecision = Chr $ (OUT_DEFAULT_PRECIS)
APIcr.FontPitchFamily = Chr $ (DEFAULT_PITCH Or FF_DONTCARE)
APIcr.FontQuality = Chr $ (DEFAULT_QUALITY)
APIcr.FontStrikeThru = Chr $ (Check1(1).Value)
APIcr.FontUnderLine = Chr $ (Check1(2).Value)
If Check1(3).Value = 1 Then APIcr.FontWidth = 20 Else APIcr.FontWidth = 0
If Check1(4).Value = 1 Then APIcr.FontWeight = 700 Else APIcr.FontWeight = 0
If Check1(5).Value = 1 Then APIcr.FontRotate = 1800 Else APIcr.FontRotate = 0
''''''''''''''''''''''''''''''''''''''

FontSelection% = CreateFontIndirect(APIcr)
Focus% = SelectObject(ArtWork.hDC, FontSelection%)
R = SetTextAlign(ArtWork.hDC, Align)
''''''''''''

R = TextOut(ArtWork.hDC, TextX, TextY, (Text1.Text), Len(Text1.Text))
R = SelectObject(ArtWork.hDC, Focus%)
R = DeleteObject(FontSelection%)
End Sub
```

APIcr.FontCharSet：FontLog 属性设置与清单 12.11 中 APICRLF_Click 事件相同。文本显示时可以从这些值中任选其一以改变文本的外观。参考清单 12.11 可了解 FontLog 项做了哪些工作。

R=TextOut：这个 API 函数 TextOut 的例子非常简单，它把包括内嵌回车符和换行符的文本绘入文本框 Text1 中。

清单 12.13 中的 ArtWork_MouseDown 事件初始化待绘文本的起点坐标。

清单 12.13　设置文本坐标

```
Sub ArtWork_MouseDown (Button%, Shift%, X As Single, Y As Single)
```

```
    ArtWork.Cls
    ArtWork.DrawWidth = 2
    ArtWork.PSet (X, Y)
    ArtWork.DrawWidth = 1
    ArtWork.CurrentX = X: ArtWork.CurrentY = Y
    TextX = X: TextY = Y
End Sub
```

ArtWork.Cls：擦除图片框中显示的文本。

ArtWork.PSet(X,Y)：画一个点以指出 CurrentX 和 CurrentY 的位置。

TextX=X:TextY=Y：保存 CurrentX 和 CurrentY 坐标值。

12.11　使用 ScaleLeft 属性放置文本

按下标有 Basic Print 的按钮时程序切换 BasicPrint_Click 事件。当 Visual Basic 的 Print 语句碰到文本中内嵌的回车符时，CurrentX 被置为 0(见清单 12.24)，这时，下一行文本就被输出在窗体或图片框最左边的地方。如果改变窗体或图片框的 ScaleLeft 属性，仍可使文本正确对齐，这只需把 0 坐标调整为第一行文本起始点的 X 位置。选项按钮单击(Click)事件中的 SetTextAlign(ArtWork.hDC,Align)语句可以按照左对齐、居中和右对齐方式对齐文本。

清单 12.14　用 Print 语句绘制文本

```
Sub BasicPrint_Click ()
    ArtWork.Cls
    ArtWork.CurrentX = TextX
    ArtWork.CurrentY = TextY
If Text1.Text > "" Then 'print attributes
  Select Case Style_Combo.Text
    Case "Normal"
        ArtWork.FontBold = False:     ArtWork.FontItalic = False
    Case "Bold"
        ArtWork.FontBold = True:      ArtWork.FontItalic = False
    Case "Italic"
        ArtWork.FontItalic = True:    ArtWork.FontBold = False
    Case "Bold-Italic"
        ArtWork.FontBold = True:      ArtWork.FontItalic = True
  End Select
ArtWork.FontName = ViewFont.FontName
ArtWork.FontSize = Val(SizeBox.Caption)
  Select Case Align      'Scale Aligning
    Case 0, 6, 2 'Left
    ArtWork.ScaleLeft = -ArtWork.CurrentX
  End Select
End If
  ArtWork.Print Text1.Text
  ArtWork.ScaleLeft = ArtWork.CurrentX 'reset ScaleLeft
End Sub
```

ArtWork.ScaleLeft=-ArtWork.CurrentX：如果给 ScaleLeft 属性赋一个负的 CurrentX 值，可使文本左对齐于 MouseDown 的 X 和 Y 坐标。BasicPrint_CLick 事件的例程与第 2 章中 Artisan 程序中的例程相同。这里的主要手段是改变图片框的 ScaleLeft 属性来设置 CurrentX，然后用 API 函数 SetTextAlign 完成文本的对齐处理。

清单 12.15 中的 FontList_Click 事件改变字体的标准、斜体和粗体属性风格，然后把字体显示于 ViewFont 标题控制对象中。

清单 12.15 设置字体的样式类型

```
Sub FontList_Click ()
ViewFont. FontName = FontList. Text
    Select Case Style_Combo. Text
        Case "Normal"
            ViewFont. FontBold = False: ViewFont. FontItalic = False
        Case "Bold"
            ViewFont. FontBold = True: ViewFont. FontItalic = False
        Case "Italic"
            ViewFont. FontItalic = True: ViewFont. FontBold = False
        Case "Bold-Italic"
            ViewFont. FontBold = True: ViewFont. FontItalic = True
    End Select
    ViewFont. Caption = FontList. Text
End Sub
```

ViewFont. FontName=FontList. Text：从字体列表中选择一个字体名时，此字体名便按照其字体样式显示于 ViewFont 标题控制对象中。

Select Case Style_Combo. Text：ViewFont 的显示与 Style_Combo 列表框中选中的字体属性（标准、斜体、粗体）相匹配。

12.12 显示 FontLog 属性

按下标有 FontLog 的按钮则切换列 FontLogBtn_Click 事件。此事件按顺序显示 APIcr 中当前存储的属性。

清单 12.16 显示控制对象中字体的当前属性

```
Sub FontLogBtn_Click ()
r = Chr(13) + Chr(10)
M$ = "Font Log values" + r + r
M$ = M$ + "FontCharSet " + Format$ (Val(APIcr. FontCharSet)) + r
M$ = M$ + "FontClipPrecision " + Format$ (Val(APIcr. FontClipPrecision)) + r
M$ = M$ + "FontHeight " + Format$ (APIcr. FontHeight) + r
M$ = M$ + "FontItalic " + Format$ (Val(APIcr. FontItalic)) + r
M$ = M$ + "FontOutPrecision " + Format$ (Val(APIcr. FontOutPrecision)) + r
M$ = M$ + "FontPitchFamily " + Format$ (Val(APIcr. FontPitchFamily)) + r
M$ = M$ + "FontQuality " + Format$ (Val(APIcr. FontQuality)) + r
M$ = M$ + "FontRotate " + Format$ (APIcr. FontRotate) + r
M$ = M$ + "FontStrikeThru " + Format$ (Val(APIcr. FontStrikeThru)) + r
M$ = M$ + "FontUnderLine " + Format$ (Val(APIcr. FontUnderLine)) + r
M$ = M$ + "FontWidth " + Format$ (APIcr. FontWidth) + r
```

```
M$ = M$ + "FontWeight " + Format$(APIcr.FontWeight) + r
M$ = M$ + "FontFaceName " + r + Format$(APIcr.FontFaceName) + r
    Text1.Text = M$
        FontLogBtn.Enabled = False
End Sub
```

Text1.Text=M$：显示字体的当前属性。

12.13　使用 API 文本对齐函数

选择一个选项按钮为 API 函数 SetTextAlign 函数设置对齐属性值(见清单 12.17)。可用 Visual Basic 的 Choose 语句代替 Select Case 模块。这样可以缩短例程。

清单 12.17　设置字体的对齐属性

```
Sub Option1_Click (Index As Integer)
   Select Case Index
     Case 0 ' Align Left
         Align = 0
     Case 1 ' Align Center
         Align = 6
     Case 2 ' Align Right
         Align = 2
End Select
    API = SetTextAlign(ArtWork.hDC, Align)
End Sub
```

API=SetTextAlign(ArtWork.hDC,Align)：设置文本的对齐调整(justification)。

清单 12.18 中的 Style_Combo_Change 事件强迫 FontList_Click 事件执行以改变字体的样式属性。

清单 12.18　设置字体的样式

```
Sub Style_Combo_Change ()
    FontList_Click
End Sub
```

FontList_Click：从 Style_Combo 列表中选择标准、斜体、粗体中的任一项便触发并执行 FontList_Click 事件(见清单 12.15)。FontList_Click 事件中包含了一个改变当前使用字体样式的例程。

12.14　显示一种字体的文本特征值(Metrics)

按下标有 TextMetrics 的按钮则程序切换至 TextMex_Click 事件。此事件按顺序显示当前字体的特征值(见清单 12.19)。

清单 12.19 显示当前字体特征值

```
Sub TextMex_Click ()
    Dim tm As TEXTMETRIC
    Dim textbuffer As String * 20
    r = Chr(13) + Chr(10)
If FontSelection% = 0 Then Exit Sub
FontSelection% = CreateFontIndirect(APIcr)
Focus% = SelectObject(ArtWork.hDC, FontSelection%)
API = GetTextMetrics(ArtWork.hDC, tm)
API = GetTextFace(ArtWork.hDC, 19, textbuffer)
M$ = "Text Metric values" + r + r
M$ = M$ + "tmHeight " + Str$(tm.tmHeight) + r
M$ = M$ + "tmAscent " + Str$(tm.tmAscent) + r
M$ = M$ + "tmDescent " + Str$(tm.tmDescent) + r
M$ = M$ + "tmInternalLeading " + Str$(tm.tmInternalLeading) + r
M$ = M$ + "tmExternalLeading " + Str$(tm.tmExternalLeading) + r
M$ = M$ + "tmAveCharWidth " + Str$(tm.tmAveCharWidth) + r
M$ = M$ + "tmMaxCharWidth " + Str$(tm.tmMaxCharWidth) + r
M$ = M$ + "tmWeight " + Str$(tm.tmWeight) + r
M$ = M$ + "tmItalic " + Str$(Val(tm.tmItalic)) + r
M$ = M$ + "tmUnderlined " + Str$(Val(tm.tmUnderlined)) + r
M$ = M$ + "tmStruckOut " + Str$(Val(tm.tmStruckOut)) + r
M$ = M$ + "tmFirstChar " + Str$(Asc(tm.tmFirstChar)) + r
M$ = M$ + "tmLastChar " + Str$(Asc(tm.tmLastChar)) + r
M$ = M$ + "tmDefaultChar " + Str$(Val(tm.tmDefaultChar)) + r
M$ = M$ + "tmBreakChar " + Str$(Val(tm.tmBreakChar)) + r
M$ = M$ + "tmPitchAndFamily " + Str$(Val(tm.tmPitchAndFamily)) + r
M$ = M$ + "tmCharSet " + Str$(Val(tm.tmCharSet)) + r
M$ = M$ + "tmOverhang " + Str$(tm.tmOverhang) + r
M$ = M$ + "tmDigitizedAspectX " + Str$(tm.tmDigitizedAspectX) + r
M$ = M$ + "tmDigitizedAspectY " + Str$(tm.tmDigitizedAspectY) + r
M$ = M$ + "Font name" + r + textbuffer
     Text1.Text = M$
         API = SelectObject(ArtWork.hDC, Focus%)
         API = DeleteObject(FontSelection%)
TextMex.Enabled = False
End Sub
```

Dim tm As TEXTMETRICS：在名为 tm 的结构变量中保存每个特征。

Dim textbuffer As String：用 API 函数 GetTextFace 保存字体名。此行不是 TEXTMETRICS 结构中的成员。

CRLF $：显示此消息需要回车符和一个换行符。

GetTextMetrics：获取当前字体的所有特征并保存入 tm 结构变量中。下列清单是 TEXTMETRIC 结构中使用的数据类型：

tm**Height**＝字符单元的高度
tm**Ascent**＝字符单元的上高(Ascent)
tm**Descent**＝字符单元的下高(Descent)
tm**InternalLeading**＝单元的内隙

tmExternalLeading＝单元的外隙
tmAveCharWidth＝平均字符宽度
tmMaxCharWidth＝最大字符宽度
tmWeight＝100-900/高亮、标准、粗体
tmItalic＝0 若非斜体特征时为 0
tmUnderLined＝0 若无下划线特征则为 0
tmStruckOut＝若非 StruckOut 特征则为 0
tmFirstChar＝字体设置的第一个特征
tmLastChar＝字体设置的最后一个特征
tmDefaultChar＝默认特征
tmBreakChar＝定义对齐方式的特征
tmPitchAndFamily＝参见 APICRLF_Click 事件(清单 12.11)
tmCharSet＝字体的特征(Character)设置
tmOverhang＝合成字体之间的间距
tmDigitizedAspectX＝字体的 X 斜度
tmDigitizedAspectY＝字体的 Y 斜度

12.15 APItext 程序小结

使用 BasicPrintBtn_Click 例程可能是 Visual Basic 中最简单的对齐文本的方法，只须按一下 Basic Print 按钮即可，BasicPrintBtn_Click 事件使用了 Visual Basic 的标准内部文本函数和一个 API 函数完成文本的左对齐、右对齐或中对齐。若想随意调整文本，只使用 Visual Basic 的文本函数就有些力不从心了。

按动两个 API Text 按钮中的任一个，都会调用 API 函数 Textout，这是最基本的 API 文本函数。绘制文本时若不需使用跳格(tabbing)、换行或其他字处理特性，则 API 函数 TextOut 是最合适的函数。就输出文本而言，Visual Basic Print 语句更好一些，但在某些格式化领域，Print 语句的功能显得不足。下面的 DrawText 程序将介绍功能更强的 API 函数 DrawText。

12.16 DrawText 程序

DrawText 程序使用 API 函数 DrawText 在窗体或图片框中放置文本。API 函数 DrawText 是一个真正的小型字处理程序，它可以模拟文本框或标题控制对象的功能。

所有 API 函数调用后都返回一个值。调用成功返回 True(非 0 值，比如－1)，调用失败时返回 0。在下面的例子里，变量 R 保存返回值：

```
R=DrawText(ArtWork.hDC,KeyBoard,-1,DT_CalCrect)
```

API 函数 DrawText 返回文本高度。遇到回车键时，可用此高度值为下一行文本留出空间。

DrawText 程序的窗体及其控制对象属性列于表 12.3 中。

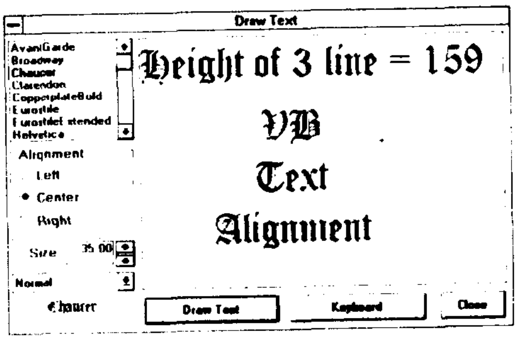

Draw Text
AvantGarde
Broadway
Chaucer
Clarendon
CopperplateBold
Eurostile
EurostileExtended
Helvetica
Alignment
Left
● Center
Right
Size
35.00
Normal
Chaucer
Height of 3 line = 159
VB
Text
Alignment
Draw Text
Keyboard
Close

图 12.5　DrawText 程序的屏幕显示

表 12.3　DrawText 窗体及其控制对象属性

Object: Form
Object Name: TextEditor
Caption　=″Draw Text″　　Height　=5040
Left　=165　　Top　=1215
Width　=7935

Object: Command Button
Object Name: Keyboard
Caption　=″Keyboard″　　Height　=420
Left　=4380　　Top　=4140
Width　=2115

Object: Command Button
Object Name: Draw_Text
Caption　=″Draw Text″　　Height　=420
Left　=2100　　Top　=4140
Width　=2115

Object: Picture Box
Object Name: Artwork
Height　=3930　　Left　=2070
MousePointer　=2 ′Cross　　ScaleMode　=3 ′Pixel
Top　=90　　Width　=5685

Object: Label
Object Name: Txt
AutoSize　=−1 ′True　　BorderStyle　=1 ′Fixed Single
Caption　=″Txt″　　Height　=225
Left　=2070　　Top　=1080
Width　=255

Object: Combo Box
Object Name: Style_Combo
Height　=300　　Left　=45
Style　=2 ′Dropdown List　　Top　=3735
Width　=1905

Object: Frame
Object Name: Frame1
Caption　=″Alignment″　　Height　=1500
Left　=45　　Top　=1710
Width　=1920

Object: Option Button
Object Name: Option1

```
Caption      ="Right"          Height       =375
Index        =2                Left         =135
Top          =990              Width        =1095
```

Object: Option Button
Object Name: Option1

```
Caption      ="Center"         Height       =375
Index        =1                Left         =135
Top          =630              Width        =1095
```

Object: Option Button
Object Name: Option1

```
Caption      ="Left"           Height       =330
Index        =0                Left         =135
Top          =315              Value        =-1 'True
Width        =1215
```

Object: Command Button
Object Name: CloseBtn

```
Caption      ="Close"          Height       =420
Left         =6720             Top          =4140
Width        =1020
```

Object: VScroll
Object Name: VScroll

```
Height       =420              LargeChange  =10
Left         =1665             Max          =4
Min          =400              Top          =3240
Value        =14               Width        =300
```

Object: List Box
Object Name: FontList

```
Height       =1590             Left         =45
Sorted       =-1 'True         Top          =90
Width        =1935
```

Object: Label
Object Name: SizeBox

```
Alignment    =1 'Right Justify BorderStyle  =1 'Fixed Single
Height       =375              Left         =900
Top          =3240             Width        =735
```

Object: Label
Object Name: ViewFont

```
Alignment    =2 'Center        AutoSize     =-1 'True
BackColor    =&H00FFFFFF&      Height       =375
Left         =45               Top          =4095
Width        =1890
```

Object: Label
Object Name: Label3

```
Alignment    =1 'Right Justify Caption      ="Size:"
FontBold     =-1 'True         Height       =255
Left         =90               Top          =3285
Width        =750
```

12.17 变量名及启动时的初始值

下列清单(清单12.20到12.23)中的事件基本上与APItext程序中用到的相同。所以在下列代码清单中,有些事件未做解释。下列控制事件的进一步解释或细节可参见APItext程序。

清单12.20 启动时用到的变量

```
Declarations
Dim Align, DT As Integer
```

Dim Align,DT:这两个变量都保存对齐方式值。

Dim BB As Rect:保存文本边界框的坐标。

清单12.21 装载字体列表

```
Sub Form_Load ()
     Style_Combo. AddItem "Normal"
     Style_Combo. AddItem "Bold"
     Style_Combo. AddItem "Italic"
     Style_Combo. AddItem "Bold-Italic"
     SizeBox. Caption = "12. 00"
     ViewFont. FontSize = 12
For R = 0 To 10
     FontNames $ = Screen. Fonts(r)
     FontList. AddItem FontNames $
Next R
     FontList. ListIndex = 0
     Style_Combo. ListIndex = 0
SizeBox. Text="14. 00"
ViewFont. FontSize=14
Txt. FontSize=14
Txt. FontName=FontList. Text
ArtWork. FontSize=14
ArtWork. FontName=FonList. Text
     ResetLabelText
End Sub
```

Form_Load:细节参见APItext程序中的Form_Load事件(见清单12.8)。

图12.6为TextDraw窗体的图例。

清单12.22 把窗体置于屏幕中间

```
Sub Form_Resize()
     TextEditor. Move(Screen. Width-Width)/2,[IC:CCC]
          (Screen. Height-Height)/2
End Sub
```

TextEditor. Move:把窗体置于屏幕中间。

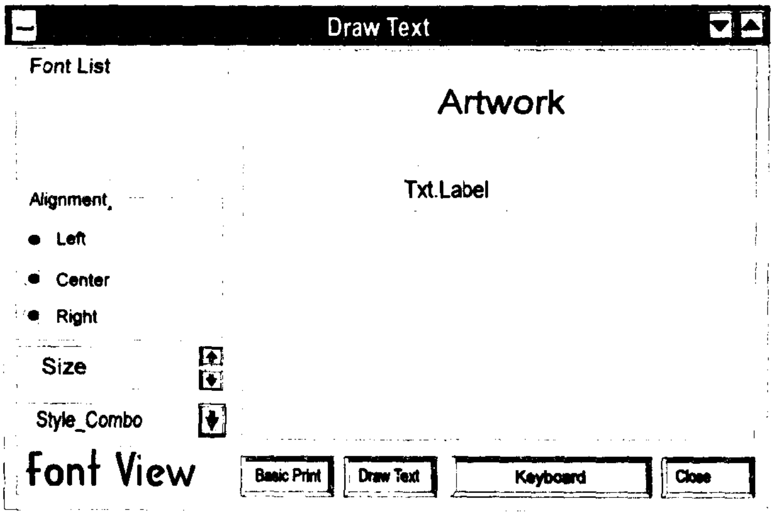

Draw Text
Font List
Artwork
Txt.Label
Alignment
● Left
● Center
● Right
Size
Style_Combo
Font View
Basic Print
Draw Text
Keyboard
Close

图 12.6　TextDraw 的窗体及其控制对象

清单 12.23　设置字体样式

```
Sub Style_Combo_Change ()
    FontList_Click
End Sub
```

FontList_Click：在 Style_Combo 列表中选择标准、斜体或粗体特性项时切换至 FontList_Click 事件（见清单 12.28）。FontList_Click 事件中包含改变当前字体样式的例程。

12.18　用 DrawText 函数绘制文本

用键盘键入字符时，API 函数 DrawText 把文本显示在图片框控制对象中。API 函数 DrawText 需要一个 Type Rect 结构把文本限制在一个边界框中。API 函数 DrawText 的 DT_CalCrect 标志用于在录入一个新字符时，自动扩大边界。

下面是可用的 DrawText 标志清单：

DT_BOTTOM：文本与边界框底边对齐（单行）。

DT_CALCREST：插入更多文本时边界框底边向下扩展（多行时）或右边向右扩展（单行时）。只处理边界框的尺寸，并无文本绘入框中。

DT_CENTER：文本居中对齐。

DT_EXPANDTABS：移动跳格键以便与正在绘制的文本对齐。默认跳格间格为 8 个字符宽。可使用 DT_TABSTOP 标志改变跳格设置。

DT_EXTERALLEADING：可用字符的外隙（External Leading）计算文本行高度。

DT_LEFT：文本左对齐。

DT_NOCLIP：文本扩展时，不裁剪边界方框。

DT_NOPREFIX：使用 & 字符时，它给下一字符加一下划线，就像 Visual Basic 中的菜单控制一样。此标志关闭这种特性。

DT_RIGHT：文本右对齐。

DT_SINGLELINE：只绘制一行文本。

DT_TABSTOP：改变跳格键的停止间隔。

DT_TOP：文本与边界框的顶部对齐（单行）

DT_VCENTER：文本在边界方框中居中显示（单行）。

DT_WORDBREAK：字符超出边界框的边界时或遇有回车符时文本切换至下一行。

键入一个字符时，ArtWork_KeyPress 事件（见清单 12.24）调用 API 函数 Drawtext，以此来模拟标题控制对象。

清单 12.24 在边界框中设置文本

```
Sub ArtWork_KeyPress(KeyAscii As Integer)
    BB.Left=Txt.Left:BB.Top=Txt.Top
    Txt.Visible=False:ArtWork.Cls

Static KeyBoard As String
KeyBoard=KeyBoard+Chr(KeyAscii)
API=DrawText(ArtWork.hDC,KeyBoard,-1,BB,DT_CalCrect)
API=DrawText(ArtWork.hDC,KeyBoard,-1,BB,DT)
End Sub
```

BB.Left,BB.Top：使边界框的左和上坐标值与标题控制对象的左和上坐标值相匹配。可使用 MouseDown 事件中的 CurrentX 和 CurrentY 设置坐标值。

KeyBoard：在 KeyBoard 变量中保存键入的所有字符。

API=DrawText #1：使用 DT_CalCrect 标志时不绘制文本而只计算边界框的尺寸。

API=DrawText #2：DT 变量中存有 DT_Alignment 标志值。对齐标志变量在 Option1_Click 事件（清单 12.30）中设置。文本随之在边界框中对齐。

清单 12.25 改变按钮的标题

```
Sub ArtWork_LostFocus ()
    KeyBoard.Caption="Keyboard"
End Sub
```

KeyBoard.Caption：Keyboard 按钮当前显示为"Type Something"，当单击任一键后，按钮标题变成"Keyboard"。

清单 12.26 在 MouseDown 位置放置标题控制对象

```
Sub ArtWork_MouseDown(Button%,Shift%,X!,Y!)
    ArtWork.Cls
    Txt.Left=X
    Txt.Top=Y
End Sub
```

Txt.Left,Txt.Top：在桌面上单击鼠标键得到的鼠标指针所指位置坐标。Txt 标题被移

入此处。

12.19　API 文本绘制和文本高度

按下标有 DrawText 的按钮时程序切换到 DrawText_Click 事件。此例程基本上与 APItext 程序中的 APICRLF_Click 事件相同，但在清单 12.27 中的 Draw_Text_Click 事件里，API 函数 DrawText 的返回值是屏幕上显示的三行文本的组合(Combined)高度。

清单 12.27　使用 DrawText 函数绘制文本

```
Sub Draw_Text_Click
    Txt.Visible=False:ArtWOrk.Cls
    ArtWork.FontSize=Val(SizeBox.Caption)
    ArtWork.FontName=ViewFont.FontName
Txt.FontSize=Val(SizeBox.Caption)
Txt.FontName=ViewFont.FontName
    BB.Left=Txt.Left
    BB.Top=Txt.Top
    BB.Right=Txt.Left+Txt.Width
    BB.Bottom=Txt.Top+Txt.Height
' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' '
Labeltext=Txt.Caption
API=DrawText(ArtWork.hDC,Labeltext,-1,BB,DT)
ArtWork.Print "Height of 3 line="+API
  ResetLabelText
End Sub
```

BB.Left=Txt.Left:API 函数 DrawText 需要一个类型为 Rect 的结构把文本包围在一个边界框中。由于标题控制对象中的文本与待提取文本相同，因此标题控制对象已有正确的尺寸大小。所以只须使 BB 等于标题控制对象的尺寸即可得到限定的尺寸。

Labeltext=Txt.Caption:获取标题控制对象中的文本。

API=DrawText(ArtWork.hDC,Labeltext,-1,BB,DT):在图片框控制对象 Artwork 中绘制文本。待绘制的文本存储在 Labeltext 中。变量-1 表示字符串中的每个字符都用到了。DT 的值等于一个对齐方式值(左对齐=&H0，居中对齐=&H1，右对齐=&H2)。

ArtWork.Print "Height of 3 Line="+API:API 函数 DrawText 返回文本高度。使用一个名为 API 的参量值调用此函数。此刻可以打印出此参量值。

清单 12.28　设置字体样式属性

```
Sub FontList_Click ()
ViewFont.FontName = FontList.Text
    Select Case Style_Combo.Text
        Case "Normal"
            ViewFont.FontBold = False: ViewFont.FontItalic = False
        Case "Bold"
            ViewFont.FontBold = True: ViewFont.FontItalic = False
        Case "Italic"
```

```
            ViewFont.FontItalic = True: ViewFont.FontBold = False
        Case "Bold-Italic"
            ViewFont.FontBold = True: ViewFont.FontItalic = True
    End Select
    ViewFont.Caption = FontList.Text
    ArtWork.FontName=FontList
    Txt.FontName=FontList
End Sub
```

FontList_Click：细节参见 APItext 程序中的 FontList 事件(见清单 12.8)。

清单 12.29 改变按钮的标题

```
Sub Keyboard_Click()
    Txt.Visible=False
    KeyBoard.Caption="Type something"
    ArtWork.SetFocus
End Sub
```

KeyBoard. Caption：Keyboard 按钮平时显示标题"Keyboard"。此时将其标题改为"TypeSomething"。

清单 12.30 设置字体的对齐方式

```
Sub Option1_Click (Index As Integer)
   Select Case Index
     Case 0 'Align Left
         Align = 0
     Case 1 'Align Centre
         Align = 6
     Case 2 'Align Right
         Align = 2
   End Select
End Sub
```

Option1_Click：细节参见 APItext 程序中的 Option1 事件(见清单 12.17)。

清单 12.31 重新设置标题控制对象中显示的文本

```
Sub ResetLabelText()
    CRLF$ =Chr(13)+Chr(10)
    Txt.Caption="VB"+CRLF$ +"Text"+CRLF$ +"Alignment
End Sub
```

Txt. Caption：Txt 的标题是"VB Text Alignment"。

12.20 DrawText 程序小结

API 函数 DrawText 有若干选项标志。这些标志用来对齐文本、设置跳格间隔、剪切显

示文本和自动处理字符的换行。TextOut 函数的返回值等于所绘文本高度。

在某些场合使用 API 文本函数可避免使用标题控制对象。配套磁盘中 16MISC 目录下的 Tabs 程序是一个很好的用 API 文本函数替代标题控制对象的例子。

12.21 本章小结

Visual Basic 中用于操纵字体特征的内部函数十分有限。本章中三个示例程序介绍了几种用手工或用 API 文本函数对齐文本的方法。

使用 API 函数可以获取行间距值(leading Value)，这既可以在纵向精确对齐文本，又可以增加或减少字符间的间隙。Visual Basic TextHeight 函数将字体设置的最大字符高度和文本字符串标准上、下行间距综合在一起。综合使用 Visual Basic 的 TextHeight 和字体的 Font Size 属性可以调整文本行的行间距。下面的例子里放入窗体的字符中首字符"T"将与窗体顶部等高，除非字体设置中还有更高的字符。不同字体中最高字符各不相同，这将影响 FontSize 和 TextHeight 的值。因此，API 文本函数可用于更精确的字体放置场合。

下面是一个 Visual Basic 行距处理方法的例子：

```
Pixely=Screen.TwipsPerPixely:CurrentlY=0
Current1Y=-(TextHeight("")-(FontSize * PixelY))/2
Print "This text is aligned at the top of the form"
```

Windows API 文本函数可简单地获取任何所需的字体属性。在改变这些字体的某些属性时，如粗体和斜体，用文本函数比用控制框属性列表的方法更好一些。因此使用 API 文本函数在这种场合是适宜的。

第 13 章　节点连接程序

节点连接程序演示如何把节点作为控制点来编辑复杂对象形状(Shape),每个节点都是一个带索引的 Visual Basic 图片框控制对象,由于可使用 Load 语句建立附加节点,故只需使用一个这样的节点图片框。在对象周围放置好所有节点后,可以利用其中任何节点改变对象的轮廓,这只要将节点图片框控制对象拖放到一个新的坐标位置即可完成。

对象编辑完毕,调用 Unload 语句消除对象中的节点,值得注意的是应使用 Load 和 Unload 语句处理节点而不应隐藏(Hide)它们,如果在每一个新形状中不断地建立节点(图片框),就很容易超过每个窗体只能有 470 个控制对象的限制。

为了便于了解连接轮廓的所有线段需用的节点总数,必须连续地对轮廓中的节点计数,并记录每个对象的起始节点和终止节点位置,以便 UpdateObjects 过程(见清单 13.17)正确地重新绘制每个形状。每个被绘对象的起始节点和终止节点的位置都用数组变量记录下来,每组多边形形状对象都有对象号码以使对象定位容易。

对每个多边形对象的每条线段都应作记录,以便将来加入一些诸如将一条或多条线段转换成曲线或者擦除某一线段的代码。加入可断开节点对象和进行 Bézier 曲线连接则要深入到程序代码内部。第 15 章的 ArtAPI 程序可将直线和曲线连接到节点上,但并不涉及可断开节点连接以及其他高级的节点特性。

13.1　程序是如何工作的

为使节点连接程序开始工作,可简单地拖放鼠标画一条线,在线的两端有一个蓝节点和一个黑节点,继续从黑节点出发画其他的线段,直到画出一个轮廓,将最后一条线段连到蓝节点上(见图 13.1)。一个带有随机填充颜色的多边形对象就形成了,参见图 13.2。

节点连接程序窗体和控制对象的属性列于表 13.1。

表 13.1　节点连接窗体及其控制对象的属性

```
Object: Form
Object Name: Main1
    Caption      ="Join & Fill"      Height      =4485
    Left         =405                Top         =825
    Width        =6855

Object: Check Box
Object Name: Main1 Check1
    Caption      ="Alternate"        Height      =195
    Left         =5265               Top         =3285
    Width        =1320

Object: Command Button
Object Name: Shapebtn
    Caption      ="Shape"            Enabled     =0 'False
    Height       =465                Left        =5220
```

Top	=3555	Width	=1365

Object: Command Button
Object Name: FloodFillBtn

Caption	="PolyFill"	Height	=375
Index	=2	Left	=3465
Top	=3285	Width	=1590

Object: Command Button
Object Name: FloodFillBtn

Caption	="ExtFloodFill"	Height	=375
Index	=1	Left	=1755
Top	=3285	Width	=1590

Object: Picture Box
Object Name: Artwork

DrawMode	=6 'Invert	Height	=3120
Left	=90	Top	=90
Width	=6540		

Object: Picture Box
Object Name: node

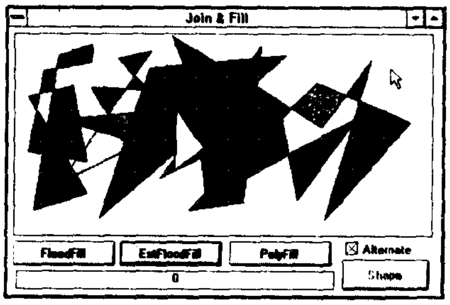
BackColor	=&H00FF0000&	BorderStyle	=0 'None
Height	90	Index	0
Left	0	ScaleHeight	90
ScaleWidth	90	TabIndex	2
Top	0	Width	90

Object: Command Button
Object Name: FloodFillBtn

Caption	"FloodFill"	Height	375
Index	0	Left	90
Top	3285	Width	1590

Object: Label
Object Name: Label1

Alignment	2 'Center	BorderStyle	1 'Fixed Singel
Height	285	Left	135
Top	3735	Width	4965



13.2 变量和初始值

节点连接程序演示如何用简单的直线构成多边形对象，每个线对象都是在其边界框内的。

清单 13.1 是节点连接程序所用变量的声明。

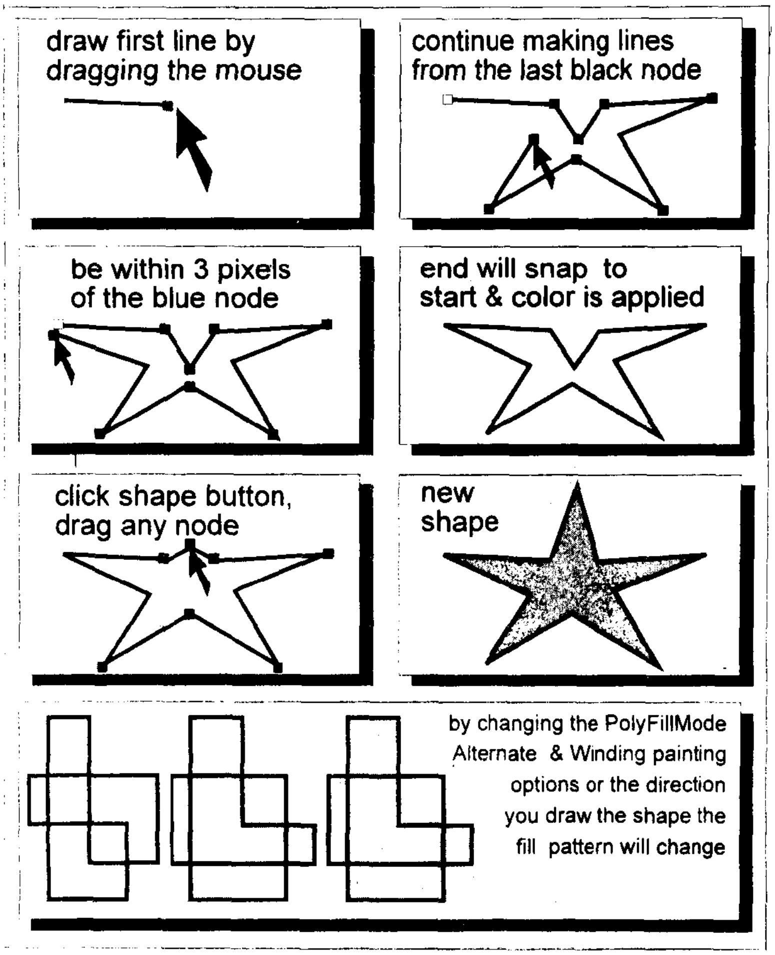

draw first line by dragging the mouse
continue making lines from the last black node
be within 3 pixels of the blue node
end will snap to start & color is applied
click shape button, drag any node
new shape
by changing the PolyFillMode Alternate & Winding painting options or the direction you draw the shape the fill pattern will change

图 13.1 线的连接

清单 13.1 变量的声明

```
Declarations
Dim X1，Y1，X2，Y2，X3，Y3 As Integer
Dim B As Integer 'Object Number
Dim Total As Integer 'Total objects
Dim PixelX，PixelY As Integer
Const Limit = 100 'Limit of objects you can draw
Dim W(Limit) As Integer 'Line Start Points
Dim N(Limit) As Integer
Dim E(Limit) As Integer 'Line End Points
Dim S(Limit) As Integer
Dim NodeTotal As Integer 'Number of nodes per object
Dim PolyShape(0 To 100) As Integer 'PolyShape number
Dim FlagPoly(Limit) As Integer 'Object Redraw Flag
Dim PolyPts As Integer 'API number of points
Dim Npt(0 To 101) As POINTAPI 'API address of each point
Dim FirstPolyPt(Limit) As Integer 'First Point of each object
Dim LastPolyPt(Limit) As Integer 'Last point of each object
Dim TotalPolyPts(Limit) As Integer 'Total points of each object
Dim FP，LP，ZZ，Counter As Integer 'calculate above to pass to API
```

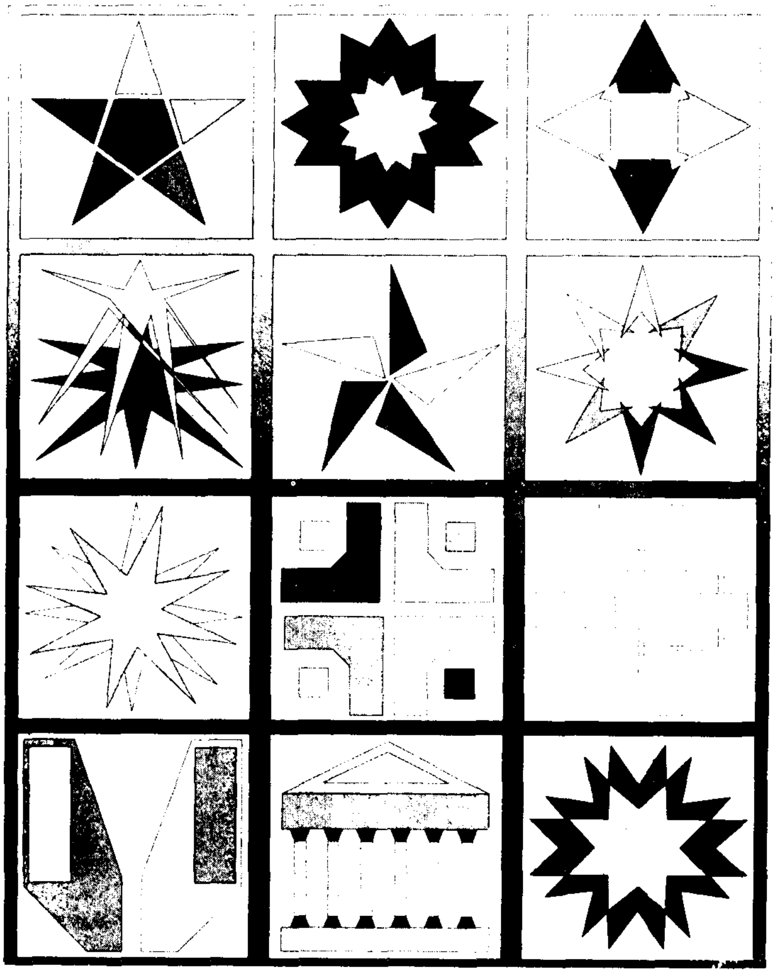


图 13.2　节点连接的连接顺序

X1,Y1,X2,Y2,X3,Y3:临时存储所画每条线的起始点和终止点,X3 和 Y3 变量用来存储连线的节点坐标。在编辑的多边形节点被移动到其他坐标重绘连线时,这些坐标是必需的。

Dim B:存储正在进行操作线段的对象号。

Dim Total:存储到目前为止所绘直线的条数。

Dim PixelX,PixelY:存放屏幕坐标每象素的 twip 转换值。若使用 twip 作为一个窗体或图片框的单位,必须转换所有的 API 多边形到象素坐标。

Const Limit=100:表示在该系统中能放置直线对象的最大数目。

Dim W,N,E,S:表示存放 X1,Y1,X2,Y2 线段起始点和终止点的四个数组(西,北,东,南)。

Dim NodeTotal:存储表示对象形状的节点总数。

Dim PolyShape:对象是由许多单独的线段组成的,因此需要把它们设置成一个多边形轮廓的直线组,以便 API 的 Polygon 函数在建立和关闭多边形时了解是哪一组线段。

Dim FlagPoly:是 UpdateObject 过程(见清单 13.17)为获知使用何种绘图函数绘制形状使用的标志,由于多边形轮廓是本程序要重绘的唯一对象类型,该标志并不是真正必

需的，留作将来使用。

Dim PolyPts：存放传送到 API 的 Polygon 函数的多边形点数。

Dim Npts：存放传送到 API 的 Polygon 函数的每个点的 X 坐标和 Y 坐标。

Dim FirstPolyPt：存放绘制对象从 Npts 数据中开始的索引号。

Dim LastPolyPt：存放绘制对象在 Npts 数据中结束的索引号。

Dim FP，LP，ZZ：临时存放多边形第一点（FP），多边形最后一点（LP）和 Dolyshape（ZZ）。

清单 13.2 开始时设定的默认值

```
Sub Form_Load ()
    PixelX = Screen. TwipsPerPixelX
    PixelY = Screen. TwipsPerPixelY
    Total = 0: B = 0
    nodeTotal = 1: Counter = 1
End Sub
```

PixelX，PixelY：把尺度模式（ScaleMode）从 Twip 转换到 Pixel，以便使 API 绘图函数正常工作。

Total＝0：B＝0：设定当前操作的直线索引号和直线对象的总数为 0。

NodeTotal＝1：使用 Load 命令增加所绘制形状必需的附加节点（图片框），为了增加附加控制对象，窗体中应有至少一个简单的已命名控制对象。由于是计算节点的个数，故在一开始 NodeTotal 就是一个在窗体上存在的节点。

Counter＝1：设置节点数为 1，即为启动时窗体上的节点数。

13.3 鼠标事件

在本程序中有两种可能的 MouseDown 情况，第一种情况是当多边形的节点可见时单击桌面（见清单 13.3），第二种情况发生在用 MouseDown 事件绘制第一条线段，以建立一个新的多边形对象（见清单 13.4）。

清单 13.3 从多边形上清除节点

```
Sub ArtWork_MouseDown (Button%, Shift%, X!, Y!)
    If NodeTotal > 1 Then
        Node(0). Visible = False
        On Error GoTo NoNode:
            For R = 1 To NodeTotal
                Unload Node(R)
            Next R
NoNode:
    UpdateObjects
    Counter = 1: ShapeBtn. Enabled = False: NodeTotal = 1
End If
ArtWork. DrawMode = 6
    If Button = 1 Then
        BoundingBox_Down Button, Shift, X, Y
        End If
    Exit Sub
    End Sub
```

If NodeTotal>1 Then：第一种情况——节点在桌面上可见。

Node(0). Visible：隐藏初始蓝节点。

On Error GoTo NoNode：准备清除在多边形对象中出现的任何节点，然而要清除那些在窗体上本不存在的节点将是个致命的错误。为安全起见，当发生错误时，程序跳至名叫 NoNode 段。

For R=1 To NodeTotal：从 1 到节点总数的所有节点。

Unload Nodes(R)：清除在 For/Next 语句中计数的所有节点。

NoNodes：On Error Goto 语句使用语句标号。

UpdateObjects：调用该过程重绘桌面上的所有多边形对象。

Counter=1：重置节点计数器为默认值 1。

ShapeBtn. Enabled=False：使形状按钮临时失效。

NodeTotal=1：将节点总数恢复为默认值 1。

ArtWork. DrawMode=6：如果 UpdateObject 过程中的某个例程改变了绘制模式，本行语句把模式复位成默认的 Inverse 模式。

If Button=1 Then：第二种情况——绘制多边形的第一条直线，只有在鼠标器左键被按下时，才执行以下模块的代码。

BoundingBox_Down：准备绘制新的直线，转入到 BoundingBox_Down 过程执行，它将绘制直线。

Exit Sub：如果清除节点时发生错误，在 On Error Goto 语句后面需有 Resume Next 语句，此处则是利用 Exit Sub 直接退出子程序。

清单 13.4 的 Artwork_MouseMove 事件调用了绘制多边形每一段的 BoundingBox_Move 子程序。

清单 13.4　绘制新的直线对象

```
Sub ArtWork_MouseMove (Button%, Shift%, X!, Y!)
    If Button = 1 Then
        BoundingBox_Move Button, Shift, X, Y
            X2 = X
            Y2 = Y
        BoundingBox_Move Button, Shift, X, Y
    End If
End Sub
```

If Button=1 Then：仅在鼠标左键被按下时继续。

BoundingBox_Move：跳至该过程，绘制一段新的直线。

X2,Y2：为体现移动而反转线的起始点和终止点的标准公式。

BoundingBox_Move：重新跳至该过程擦除已绘制的第一条直线。

清单 13.5　设置新的直线对象

```
Sub ArtWork_MouseUp (Button%, Shift%, X!, Y!)
    If MoveMouse() = True Then
        BoundingBox_Up Button, Shift, X, Y
    End If
End Sub
```

If MoveMouse()=True Then:只有鼠标移动了 MoveMouse 函数(见清单 13.10)设定的距离,才将该位置设置成直线对象最终位置,MoveMouse 函数的当前设置是 45 twip 或 3 个象素(包括任何方向)。

13.4　绘制折线段

所有的直线都是通过以下三个过程绘制的,参见清单 13.6 至清单 13.8。

清单 13.6　初始化直线的起始点和终止点坐标

```
Sub BoundingBox_Down (Button%, Shift%, X!, Y!)
    X1 = X
    Y1 = Y
    X2 = X1
    Y2 = Y1
End Sub
```

X1,Y1,X2,Y2:设置直线的起始点和终止点为 MouseDown 的当前位置。

清单 13.7　绘制直线或多段直线

```
Sub BoundingBox_Move (Button%, Shift%, X!, Y!)
    If MoveMouse() = False Then Exit Sub
        If Button Then
            ArtWork.Line (X1, Y1)-(X2, Y2)
        End If
End Sub
```

If MouseMove()=False Then Exit Sub:鼠标未移动到 MouseMove 函数设定的距离,当前距离设定为 45 twip 或 3 个象素(包括任何方向)。

If Button then:只有鼠标键被按下时才继续。

ArtWork.Line:绘制直线或折线段。

清单 13.8　多边形的自动连接

```
Sub BoundingBox_Up (Button%, Shift%, X!, Y!)
    Total = Total + 1: B = Total
    W(B) = X1: N(B) = Y1: E(B) = X2: S(B) = Y2
If Counter = 1 Then FirstPolyPt(B) = B: FlagPoly(B) = B: ZZ = B
    Snap = 120 'Distance from mouse pointer to blue node
If X2 > Node(0).Left - Snap And X2 < Node(0).Left + 90 + Snap [IC:CCC]
```

```
And Y2 > Node(0).Top - Snap And Y2 < Node(0).Top + 90 + Snap [IC:CCC]
Then E(B) = Node(0).Left + 45: S(B) = Node(0).Top + 45
    For r = 1 To NodeTotal - 1
        Unload Node(r)
    Next r
    NodeTotal = 1: LastPolyPt(ZZ) = B: TotalPolyPts(ZZ) = Counter
    PolyShape(B) = ZZ: ShapeBtn.Enabled = True
    UpdateObjects
End If
    Nodes 'Load new nodes
End Sub
```

Total＝Total＋1：若鼠标移动了 45 twip，窗体中将绘制一条直线，total 加 1 以表示这个新的直线对象。

B：分配一个号码给直线，此处它等于 Total。

W(B),N(B),E(B),S(B)：存放起始点(X1,Y1)和终止点(X2,Y2)坐标到索引数组中，以便将来使用。

If Counter＝1：计数器记录加到对象上的每一个节点，第一个节点的节点坐标需传至 API 函数 Polygon。

FlagPoly＝B：为使 UpdateObject 过程了解使用何种绘制函数，应存储所绘制对象的类型到该变量中。目前本程序只有一种类型的绘制动作，这就是多边对象绘制。

ZZ＝B：在 ZZ 中临时存放直线号，每增加一段直线，ZZ 将被加 1 直至最后一段直线连接到第一段直线形成一个多边形为止。

Snap＝120：可将最后一段直线的节点与第一段的节点自动连接在一起，为此需将两节点在自动连结之前的最大距离设定好。当前距离设为 120 twip 或者 8 象素(任意方向)，可随时改动这个数字以满足需要。

If X2>Node(0). Left－Snap And：如果第一个节点和最后的节点距离在 120 twip 以内，连接它们构成一个多边形。

E(B)＝Node(0). Left＋45:S(B)＝Node(0). Top＋45:E(B)和 S(B)值与最后一段线节点相等，可简单地连接该节点中心和第一节点中心(这往往就是蓝节点)。

For R＝1 To NodeTotal－1：从 1 到节点总数 Nodetotal－1 的所有节点，执行以下代码块。

Unload Node(R)：去掉每一线段上出现的所有节点。

NodeTotal＝1：重置节点总数为默认值 1。

LastPolyPts(ZZ)＝Counter：保存该多边形上点的总数以便传送到 API 的 Polygon 函数。

PolyShape(B)＝ZZ：多边形组号与绘制的最后一点相同。

UpdateObjects：重绘在桌面上出现的所有多边形。

Nodes：每个多边形均重绘后，调用 Nodes 过程隐藏蓝节点。

13.5　复杂对象的填色

使用清单 13.9 的 FloodFillBtn_Click 事件给所有的多边形填色，用鼠标箭头在一些区

域上单击然后选择 FloodFill 或者 ExtFloodFill 按钮，就可以给这些单独的区域填色。

清单 13.9 三种绘制（填充）方法

```
Sub FloodFillBtn_Click (Index As Integer)
    ArtWork.DrawMode = 13
    ArtWork.FillStyle = 0
    ArtWork.FillColor = QBColor(15 * Rnd)
If ArtWork.FillColor = QBColor(0) [IC:CCC]
            Then ArtWork.FillColor = QBColor(1)
    X = X1 \ PixelX: Y = Y1 \ PixelY
Select Case Index
     Case 0 'FloodFill
        r = FloodFill(ArtWork.hDC, X, Y, &H8)
     Case 1 'ExtFloodFill
        r = ExtFloodFill(ArtWork.hDC, X, Y, &H8, FLOODFILLBORDER)
     Case 2 'PolyFillMode
        If Check1.Value = 1 Then PolyMode = 1 Else PolyMode = 2
        r = SetPolyFillMode(ArtWork.hDC, WINDING)
        r = Polygon(ArtWork.hDC, Npt(FP), PolyPts)
End Select
End Sub
```

ArtWork.DrawMode=13：欲对一个形状或区域填色，必须改变当前的绘线模式，从反置(Inverse)改为正常(Normal)，即 copy_pen 13。

ArtWork.FillStyle=0：首先选择填充方案，此处只有 0 或 Solid。

ArtWork.FillColor：这一段短程序未使用调色板，故可选择颜色，此处使用随机选择的颜色(从 0 到 15 随机选取的一种颜色)。

If ArtWork.FillColor=0：黑颜色，它不是个好的颜色例子，改变它。

X=X1\PixelX:Y=Y1\PixelY：欲通过 FloodFill 或 ExtFloodFill 按钮给窗体的选择区域填色，必须先改变 X 和 Y 的尺度，大部分 API 绘制函数均使用象素坐标值来工作。X1 与 Y1 是 MouseDown 的当前位置 X 和 Y，PixelX 和 PixelY 把 twip 转换成象素。

Select Case Index：根据选择的按钮，完成与选择的按钮号相匹配的 Case 表达式。

Case 0：选择了 FloodFill 按钮。

```
FloodFill(hDC,X,Y,Color)
```

FloodFill=(绘制对象，开始绘制的 X，Y 屏幕位置和绘制边界使用的颜色)。

Case 1：选择了 ExtFloodFill 按钮。

```
ExtFloodFill(hDC,X,Y,Color,Option)
```

ExtFloodFill=(绘制对象，开始绘制的 X，Y 屏幕位置和绘制边界使用的颜色或绘制的象素颜色，两个绘制选项中的一个)。

Option 1: FLOODFILLBORDER：一直绘制到发现边界颜色为止，与 FloodFill 函数相同。

Option 2:FLOODFILLSURFACE：仅在设定颜色之上绘制，直到绘完周围颜色相同的

象素为止。

Case 2：选择了 PolyFill 按钮。

```
SetPolyFillMode(hDC,Option)
```

SetPolyFillMode＝(要设置的对象及两个选项中的一个)。

Option 1：ALTERNATE：从左到右每隔一个闭合区域进行填色。

Option 2：WINDING：根据对象的每段的绘制方向决定填充的方式，参见图 13.2。

清单 13.10 中的 MoveMouse 函数判断在鼠标拖动中鼠标是否移动了给定的距离。

清单 13.10　判断鼠标是否移动了给定的距离

```
Function MoveMouse ()
    If X1 > X2 + 45 Or X1 < X2 - 45 Or Y1 > Y2 + 45 Or [IC:CCC]
            Y1 < Y2 - 45 Then
        MoveMouse = True
    Else
        MoveMouse = False
    End If
End Function
```

If X1>X2+45 Or etc：用户单击鼠标时，激活了 MouseDown 事件，X1，Y1，X2 和 Y2 与鼠标的 X 和 Y 坐标等值，可给该位置加上一个值以判断鼠标是否移动了给定的距离，此处距离为 45 twip，也就是在任何方向上均为 3 个象素。可改变该值以满足需要。

13.6　使用节点编辑多边形

对象中节点的顺序是由加载时指定给它的序号决定的，节点连接程序视 Node(1)为多边形第一条线段的节点，最初的 Node(0)则隐藏在 Node(1)中。

在第 15 章的 ArtAPI 程序中，第一条线段的节点是 Node(0)，然后按顺序增加。因而在使用时必须决定第一条线段上第一个节点的开始序号(参见清单 13.11)。在开始增加更多复杂段和编辑功能时，你会发现一个例程比另一个更合适。

清单 13.11　设置附加线段的起始点

```
Sub node MouseDown (Index%, Button%, Shift%, X!, Y!)
    ArtWork.DrawMode = 6
    X1 = Node(Index).Left + 45
    Y1 = Node(Index).Top + 45
    X2 = X1
    Y2 = Y1
End Sub
```

ArtWork. DrawMode＝6：若绘制模式已改变，重置为反转(Inverse)模式。

X1，Y1：设置直线的起始坐标为前一节点的中心。

X2，Y2：从前一节点中心开始绘线。

清单 13.12 中的 Node_MouseMove 事件在节点移动时绘制两段与该节点相连的线段。

清单 13.12　移动多边形的线段

```
Sub node_MouseMove (Index%, Button%, Shift%, X!, Y!)
Label1.Caption = Format $ (Index)
    If Button = 1 Then
        If ShapeBtn.Enabled = True Then NodeMove Index,[IC:CCC]
             Button, Shift, X, Y: Exit Sub
    BoundingBox_Move Button, Shift, X, Y
        X2 = X
        Y2 = Y
    BoundingBox_Move Button, Shift, X, Y
End If
End Sub
```

Label1.Caption：当鼠标在节点上通过时，该标志显示分配给该节点的号码，此时节点号 1 定位在节点号 0 之下。

If Button=1 Then：仅当鼠标键被按下才继续。

If ShapeBtn.Enabled：只有在多边形完成后形状按钮(Shape Button)才被激活，此时方可进入 NodeMove 子程序(见清单 13.14)编辑刚绘制好的对象轮廓。

BoundingBox_Move：如果形状按钮是禁止，这意味着你仍在绘制对象的线段，BoundingBox_Move 子程序(见清单 13.7)从此处接管过去。

X2,Y2：反转直线的终止点或起始点。

BoundingBox_Move：重复子程序以擦除已绘制的第一条直线。

清单 13.13 中的 Node_MouseUp 事件在多边形被编辑后重新计算节点的坐标，然后调用 UpdateObjects 子程序重绘多边形。

清单 13.13　设置多边形的每一条新线段

```
Sub node_MouseUp (Index%, Button%, Shift%, X!, Y!)
    If ShapeBtn.Enabled = True Then
        Node(Index).Left = X - 45: Node(Index).Top = Y - 45
        Node(Index).ScaleLeft = X - 45: Node(Index).ScaleTop = Y - 45
        Node(1).Left = Node(0).Left: Node(1).Top = Node(0).Top
        Node(1).ScaleLeft = Node(0).ScaleLeft:
        Node(1).ScaleTop = Node(0).ScaleTop
            UpdateObjects
    Exit Sub
End If
    ArtWork_MouseUp Button, Shift, X, Y
End Sub
```

If ShapeBtn.Enabled=True Then：多边形对象绘制完成后该按钮才被激活，这给移动到新的屏幕位置的节点重新定位。与第 2 章的移动直线时拖动节点(Shape 控制对象)的 Artisan 程序不同，本程序采用的是绘制类似于节点控制对象的方框。

Node(1).Left=Node(0).Left:Shape 按钮被按下后,放置第一个节点到蓝节点之下时才使用本行语句,为节省代码空间放在此处。蓝节点 Node(0)在闭合的多边形中用作多边形的终止节点,Node(1)用作多边形的起始节点,在节点连接处打破多边形时,可将蓝节点跳至断点处再次用作终止节点。本例中节点打破后没有恢复。

Node(1).ScaleLeft=Node(0).ScaleLeft:移动一个节点到新的位置时,必须对内部修改比例,以反映 Artwork 图片框桌面的比例尺度。

UpdateObjects:重绘目前显示的所有的对象并退出该事件。

Artwork_MouseUp:如果 Shape 按钮禁止,却绘制了线段,这就通过强迫的 Artwork 图片框的 MouseUp 事件自动绘制了一段线。

13.7　移动连接的线段

移动一个节点时,与之相连的两条直线必然同时而移动,可以使用清单 13.14 中的 NodeMove 子程序同时移动相连的两条线,为安全地重绘所有的多边形,需要一个公式来查找节点 X 和 Y 所存储的序号。

清单 13.14　编辑多边形的节点位置

```
Sub NodeMove (Index%, Button%, Shift%, X!, Y!)
    If Button Then
        Z = PolyShape(B)
        r = FirstPolyPt(Z) + Index - 2
        rr = FirstPolyPt(Z) + Index
If rr > LastPolyPt(Z) Then rr = FirstPolyPt(Z)
If Index = 0 Then r = FirstPolyPt(Z) + 1: rr = LastPolyPt(Z)
    X1 = W(r): Y1 = N(r): X3 = W(rr): Y3 = N(rr)
ArtWork.Line (X1, Y1)-(X2, Y2): ArtWork.Line -(X3, Y3)
ArtWork.Line (X2 - 45, Y2 - 45)-(X2 + 45, Y2 + 45), , B
    X2 = X
    Y2 = Y
ArtWork.Line (X1, Y1)-(X2, Y2): ArtWork.Line -(X3, Y3)
ArtWork.Line (X2 - 45, Y2 - 45)-(X2 + 45, Y2 + 45), , B
    If Index = 0 Then
        W(r - 1) = X: N(r - 1) = Y
    Else
        W(r + 1) = X: N(r + 1) = Y
    End If
End If
End Sub
```

If Button Then:仅当鼠标键被按下才继续。

Z=PolyShape(B):B 的值等于最后一条线段的号码,与 B 值对应的 PolyShape()序列号码指示组成多边形线段组的第一个坐标序号。

r=FirstPolyPt(z)+Index-2:在本程序中蓝节点通常序号为 0,紧随其后的节点序号为 2,3,4……由于节点 1(Node(1))隐藏在蓝节点下,这使得计算过程有点技巧性。移动时必需了解该节点两侧的节点,变量 r 给出了移动节点的下一个最低节点。

rr=FirstPolyPt(z)+Index：需同时移动两条线段，变量 rr 给出了移动节点的下一个最高节点。

If rr>LastPolyPt(z) Then：多边形的最后节点带出一个问题，下一个最高节点是封闭多边形中的第一节点。在移动节点链中最后一个节点时，本语句负责应有的变更。

If Index =0 Then：多边形的第一节点也有类似的问题，下一个最低节点实际上是封闭多边形的最后节点。在移动第一节点时，本语句负责应有的变更。

X1=W(r):Y1=N(r)：使前一节点与欲移动节点相等。

X3=W(rr):Y3=N(rr)：使后续节点与欲移动节点相等。

ArtWork. Line(X1,Y1)-(X2,Y2):ArtWork. Line-(X3,Y3)：绘制两条相连的线。

ArtWork. Line(X2-45,Y2-45)-(X2+45,Y2+45),,B：绘制一个黑色节点。

X2=X:Y2=Y：反转可移直线和节点的描述值。

ArtWork. Line：重绘两条直线和一个黑色节点，擦除已绘制的第一组。

If Index=0 Then W(r-1)=X:N(r-1)=Y Else w(r+1)=X:N(r+1)=Y：最后，存储新线段的开始点到合适的序号位置。

13.8 增加更多的图片框节点

清单 13.15 的 Nodes 过程简单地给绘制的新的直线增加新的节点，每个节点都是个图片框控制。为了在运行期间增加新的控制对象，可以使用 Load 语句，加上控制对象的名字和分配给它的序号，在装载附加控制对象之前在窗体内必须至少有一个同样名字的控制对象，装载新的节点后，把它重放在前一线段的终点，并给节点内部重定比例，以反映 Artwork 图片框桌面的比例。

清单 13.15 装载所需的新节点

```
Sub Nodes ()
    If Counter = 1 Then 'First blue node
        Node(0).Visible = False
        Node(0).Left = W(B) - 45: Node(0).Top = N(B) - 45
        Node(0).ScaleLeft = Node(0).Left:
        Node(0).ScaleTop = Node(0).Top
        Node(0).Visible = True
    End If
r = NodeTotal 'add new node
Load Node(r): Node(r).BackColor = &HF&
    Node(r).Left = E(B) - 45: Node(r).Top = S(B) - 45
    Node(r).ScaleLeft = Node(r).Left:
    Node(r).ScaleTop = Node(r).Top
NodeTotal = NodeTotal + 1
Counter = Counter + 1
    Node(r).Visible = True
End Sub
```

If Counter=1：如果 Counter 变量等于 1，表示正在绘制多边形对象的第一条线段，该线隐藏了第一个节点(蓝色)，然后移动它到第一条线的起始点，移动后该节点再现。

NodeTotal＝NodeTotal＋1：节点总数和计数器加 1，下一个 r 变量等于下一节点的序列号码。

Load Node(r)：Node(r). BackColor＝&HF&：一个新的图片框控制对象装载到窗体上，使节点变黑，修改比例并重放它到绘制的最后一条线的末端。

Node(r). Visible＝True：必须使新控制对象可见。所有的新控制对象在第一次装载时都是默认为不可见的。

13. 9　装载对象的所有节点

按下轮廓按钮时，它将装载所有节点(图片框)到新绘制的多边形上，参见清单 13. 16。

清单 13. 16　放置多边形周围的节点

```
Sub ShapeBtn_Click ()
    If nodetotal > 2 Then Exit Sub
    g = PolyShape(B): rr = 2
    For r = FirstPolyPt(g) + 1 To LastPolyPt(g)'add new nodes
        Load Node(rr): Node(rr).BackColor = BLACK
        Node(rr).Left = W(r) - 45: Node(rr).Top = N(r) - 45
        Node(rr).ScaleLeft = Node(rr).Left:
        Node(rr).ScaleTop = Node(rr).Top
        Node(rr).Visible = True: rr = rr + 1
    Next r
    NodeTotal = rr - 1: Node(0).Visible = True
End Sub
```

g＝PolyShape(B)：rr＝2：B 变量存放着最后绘制线段的序号，Polyshape()数组存放着传送到 FirstPolyPt 和 LastPolyPt 数组的相应序号，凭它们找到多边形轮廓的第一和最后节点位置。

rr＝2：和设置给第一个新节点的序号相等，Node(0)的序列号 0 和 Node(1)的序列号 1 是不会被卸载的，从序列号 2 开始装载节点。

For R＝FirstPolyPt(g)＋1 To LastPolyPt(g)：从第一条线段的起始节点到最后一条线段的节点，完成下面的代码块。

Load Node(rr)：装载一个新的节点图片框控制对象。

Node(rr). BackColor＝BLACK：初始的节点 Node(0)是蓝色的，若不改变每个新节点的 BackColor，它也会是蓝色的。

Node(rr). Left：放置每个新节点到形状的适当线段的起始端。

Node(rr). ScaleLeft：重新给每个新节点定比例，以便它的内部比例反映在 Artwork 桌面上。

Mode(rr). Visible＝True：运行期间增加的新控制对象的默认值为不可见。

rr＝rr＋1：序号加 1。

NodeTotal＝rr－1：存储目前在对象周围可见的节点数。

Node(0). Visible＝True：使蓝节点重新可见。

13.10 绘制最终的多边形

清单 13.17 是第 5 章 Artisan 程序和第 15 章 ArtAPI 程序的 UpdateObject 过程的简化版本，所有的 UpdateObject 过程都绘制最终的对象形状。

清单 13.17 重绘多边形

```
Sub UpdateObjects ()
    ArtWork.Cls: LastB = B: B = 1
Do While B < Total + 1
    Select Case B
        Case FlagPoly(B)
            FP = FirstPolyPt(B): LP = LastPolyPt(B)
            Npt(1).X = W(FP) \ PixelX: Npt(1).Y = N(FP) \ PixelY
    For BB = FP To LP
        Npt(BB).X = W(BB) \ PixelX: Npt(BB).Y = N(BB) \ PixelY
    Next BB
        PolyPts = TotalPolyPts(B): Counter = 0
        FloodFillBtn_Click 2
End Select
    B = B + 1
Loop
    B = LastB
End Sub
```

ArtWork.Cls：清屏并消除所有绘制的对象。

LastB=B:B=1：临时存储最后线段号码并令 B 为 1。

Do While B<Total+1：连续执行以下语句模块，直到所有的线段都被重绘到 ArtWork 桌面上。

Select Case B：跳至与标号相应的绘制函数，本程序中仅有一种标号。

Case FlagPoly(B)：重绘到目前为止所有的多边形轮廓。

API 的 Polygon 函数如下：

```
Polygon(ArtWork.hDC,Npt(FP),PolyPts)
```

FP=FirstPolyPt(B)：表示查找多边形的第一点 X,Y 坐标的序列号。

LP=LastPolyPt(B)：表示查找多边形的最后点 X,Y 坐标的序列号。

Npt(1).X=W(FP)\PixelX：设置多边形的第一点 X,Y 坐标，类似于用 Pset 语句开始一个 step 语句行。由于 API 函数 Polygon 使用象素模式，转换 twip 值到象素值。

For BB=FP To LP：从第一条线段到最后一条的数组循环。

Npt(BB).X=W(BB)\PielX：转换每一个附加的 X 和 Y 值到象素坐标。

PolyPts=TotalPolyPts(B)：传送到 API 的 Polygon 函数的值，指示绘制在多边形上相应的总数。

FloodFillBtn_Click 2：强迫一个 FloodFillBtn 事件，以逐一绘制多边形，参数 2 引起 Select Case Index 2 的例程的执行。

13.11　节点连接程序中的API函数

清单13.18列出了Type结构和在节点连接程序中使用的API绘图函数。

清单13.18　节点连接程序中使用的API函数

```
Bas Module file declarations.
Type POINTAPI
    X As Integer
    Y As Integer
End Type
Declare Function FloodFill% Lib "GDI" (ByVal hDC As Integer, [IC:CCC]
ByVal X As Integer, ByVal Y As Integer, ByVal crColor As Long)
Declare Function ExtFloodFill% Lib "GDI" (ByVal hDC As Integer, [IC:CCC]
ByVal X As Integer, ByVal Y As Integer, ByVal crColor As Long, [IC:CCC]
ByVal wFillType As Integer)
Declare Function SetPolyFillMode% Lib "GDI" (ByVal hDC As Integer, [IC:CCC]
Integer ByVal nPolyFillMode As Integer)
Declare Function Polygon% Lib "GDI" (ByVal hDC As Integer, [IC:CCC]
lppoints As POINTAPI, ByVal nCount As Integer)
Global Const FLOODFILLSURFACE = 1
Global Const FLOODFILLBORDER = 0
Global Const ALTERNATE = 1
Global Const WINDING = 2
Global Const BLACK = &H0&
Global Const RED = &HFF&
Global Const GREEN = &HFF00&
Global Const YELLOW = &HFFFF&
Global Const BLUE = &HFF0000
Global Const MAGENTA = &HFF00FF
Global Const CYAN = &HFFFF00
Global Const WHITE = &HFFFFFF
These are the Windows API functions that the Join Nodes program uses.
FloodFill (hDC,X,Y,Color)
ExtFloodFill(hDC,X,Y,Color,FillType)
SetPolyFillMode(hDC,PolyFillMode) ALTERNATE=1;WINDING=2
Polygon(hDC,POINTAPI,Count)
```

13.12　本章小结

节点连接程序显示了将线对象连接在一起的基本结构，为存放和查找多边形对象中的节点，可以使用多个数组结构变量，有些程序使用了类似于3-D模型化的数据结构来存放和查找节点。必须决定存放和提取节点坐标的方法，如果需要编写复杂的图形编辑程序，这种决定也是很重要的。

在第15章“ArtAPI程序”中将使用本章所使用的技术来连接直线对象和曲线，这种过程有时会变得很复杂，因而读者应该在节点连接程序中完全理解这种数组结构。

即使不需要在程序中存在可编辑的连接直线段和曲线段，仍有可能需要建立本章程序的框架来增强其他程序的功能。一种可能的用途是利用鼠标直接进行编辑的直线图表，另一种可能的用途是建立的2-D位图动画中的路径连线。

第 14 章　定标与打印程序

Visual Basic 为打印图形和测试打印机提供两种方法：打印机对象(Printer Object)法和打印窗体(PrintForm)法。打印机对象法直接向打印机发送 Circle，Line，Print 和 Pset 语句(和向窗体或图片框发送对象相似)，图形和文本对象发送到打印机之后，用 EndDoc 语句打印当前页，Visual Basic 关键字 NewPage 可用以打印多页。打印窗体法是对窗体内容作硬拷贝，然后以显示器分辨率打印硬拷贝图形，由于大部分显示器分辨率均低于高级打印机，故只能打印出令人失望的图象。

可同时发送 API 图形和文本函数至打印机，为正确放置 API 图形和文本位置，在某种情况下，有必要作附加的定标运算。

Windows API 图形函数要求传送以象素为单位的坐标，PrintAPI 程序的 Art 图片框则使用 twip 单位作为其定标模式，所有内部图片框坐标在传送至 API 图形函数处理之前必须从 twip 转换成象素单位。

PrintAPI 程序在图片框控制对象上绘制两个框、一个圆和一条对象线对象，其较大框对象代表打印页上可打印的空白区域，可使用此程序测试两个设备(图片框和打印页)图象区的定标设置，并进行人工设置。

为正确地通过 API 函数向打印机定标一幅图象，必须计算 API 象素分辨率与打印机 DPI 分辨率间的区别，以下为转换 API 象素分辨率至打印机分辨率的两个公式例子：

1. Npt(Z).X=(Npt(Z).X * HPrinterDPI)−LeftM
   Npt(Z).Y=(Npt(Z).Y * VPrinterDPI)−TopM
2. Npt(Z).X=(Npt(Z).X−LeftM) * HPrinterDPI
   Npt(Z).Y=(Npt(Z).Y−TopM) * VPrinterDPI

Npt 数组存放传送至 API 图形函数的 X 坐标。

HPrinterDPI 与 VPrinterDPI 是表示每英寸上打印机象素值的变量，若在窗体或图片框中使用 twip 作为定标模式，必须将这些值转换成象素单位(参看清单 14.5 的 Print_API_Click 事件)。

用 HPrinterDPI 与 VPrinterDPI 值乘上两设备间水平宽度与垂直高度定标差。

Leftm 是表示打印页的图象区左间距。在使用第一条公式时，LeftM 是控制对象从(0,0)坐标加上的偏移量。若使用第二条公式，LeftM 是(0,0)坐标与可正可负的控制对象 ScaleLeft 属性间的差，本例图片框控制对象的 ScaleLeft 为零，因而 LeftM 是 0。

使用 TopM 与 VPrinterDPI 值计算 Npt(Z).Y 坐标，用 VPrinterDPI 值乘上两设备间垂直高度之差。

图 14.1 示出了 PrintAPI 的窗体。

PrintAPI 程序窗体和控制对象的属性列于表 14.1。

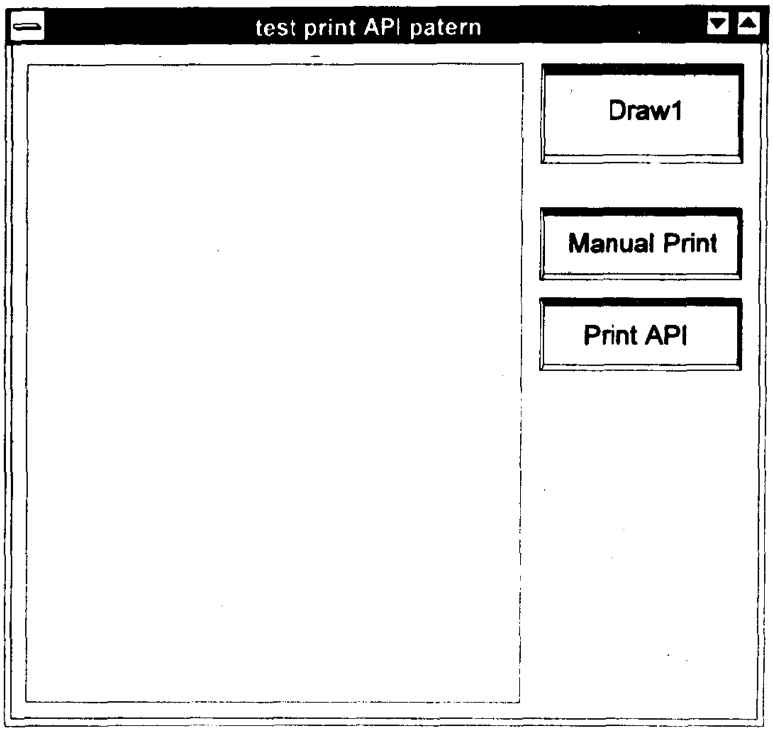


图 14.1　PrintAPI 窗体及其控制对象

表 14.1　PrintAPI 窗体及其控制对象的属性

```
Object: Form
Object Name: Form1
    BackColor     =&H00FF0000&              Left        =390
    Caption       ="test print API pattern" Top         =1185
    Height        =5865                     Width       =6255

Object: Command Button
Object Name: Print_API
    Caption       ="PrintAPI"               Left        =4380
    Enabled       =0 'False                 Top         =2040
    Height        =615                      Width       =1635

Object: Command Button
Object Name: Manual_print
    Caption       ="Manual Print"           Left        =4380
    Enabled       =0 'False                 Top         =1320
    Height        =615                      Width       =1680

Object: Command Button
Object Name: Draw1
    Caption       ="Draw1"                  Top         60
    Height        =960                      Width       =1725
    Left          =4365

Object: Picture Box
Object Name: Art
    BorderStyle   =0 None                   ScaleWidth  =4095
    Height        =5285                     Top         =120
    Left          =120                      Width       =4090
```

```
ScaleHeight      =5280
```

图 14.2 示出了 PrintAPI 程序在屏幕上的显示图象。

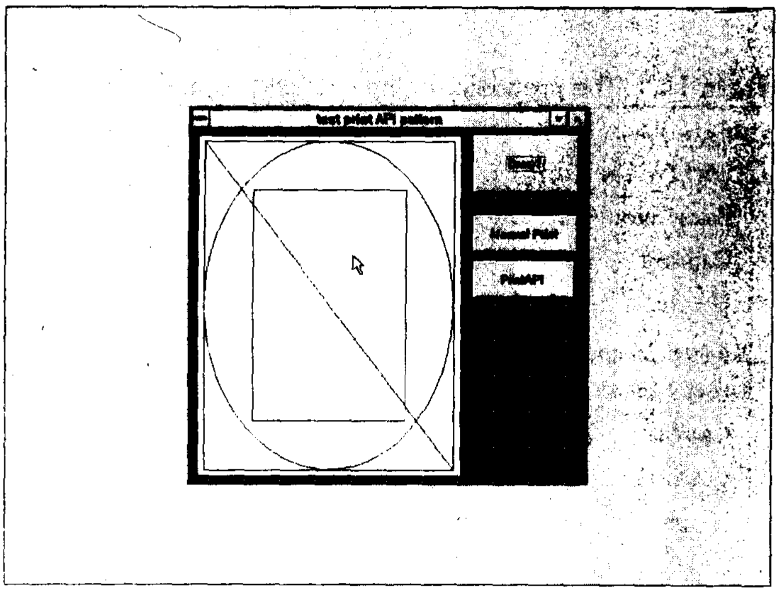


图 14.2 PrintAPI 程序的一个屏幕显示

14.1 变量名和 API 函数

PrintAPI 程序设定成向 300 dpi 打印机打印 8.5 * 11 英寸信纸页，如果打印机不提供这些默认值，则必须手工修改定标值，清单 14.1 声明部分包括了增加打印性能的额外 API 函数，打印机驱动程序控制着打印 Windows 图形命令的控制代码。几乎所有的激光打印机都能打印 PrintAPI 程序的测试页，一些点阵打印机和喷墨打印机在打印驱动程序设计成使用 Windows 3.0 或更高版本的图形时也能打印测试页。

清单 14.1 测试 API 图形函数

```
Declaration:
Declare Function EscapeBynum% Lib "GDI" Alias "Escape"
    (ByVal hDC%, ByVal nEscape%, ByVal nCount%,
    ByVal lpInData&, ByVal lpOutData&)
Declare Function DPtoLP Lib "GDI" (ByVal hDC As Integer,
    lpPoints As POINTAPI, ByVal nCount As Integer) As Integer
Declare Function CreateDC Lib "GDI"
    (ByVal lpDriverName As String, ByVal lpDeviceName As String,
    ByVal lpOutput As String, ByVal lpInitData As String)
Declare Function DeleteDC% Lib "GDI" (ByVal hDC As Integer)
Declare Function CreateMetaFile% Lib "GDI" (lpString As Any)
Declare Function LPtoDP% Lib "GDI"
    (ByVal hDC As Integer, lpPoints As POINTAPI,
    ByVal nCount As Integer)
Declare Function IsRectEmpty% Lib "User" (lpRect As Rect)
Declare Function Polygon% Lib "GDI"
    (ByVal hDC As Integer, lpPoints As POINTAPI,
    ByVal nCount As Integer)
Declare Function Polyline% Lib "GDI"
    (ByVal hDC As Integer, lpPoints As POINTAPI,
    ByVal nCount As Integer)
Declare Function SetPolyFillMode% Lib "GDI"
    (ByVal hDC As Integer, ByVal nPolyFillMode As Integer)
Declare Function Rectangle% Lib "GDI"
    (ByVal hDC As Integer, ByVal X1 As Integer, ByVal Y1 As Integer,
    ByVal X2 As Integer, ByVal Y2 As Integer)
Declare Function Ellipse% Lib "GDI"
    (ByVal hDC As Integer, ByVal X1 As Integer, ByVal Y1 As Integer,
    ByVal X2 As Integer, ByVal Y2 As Integer)
Const HORZRES = 8: Const VERTRES = 10
Declare Function GetDeviceCaps% Lib "GDI"
    (ByVal hDC As Integer, ByVal nIndex As Integer)
Dim HPrinterDPI As Single
Dim VPrinterDPI As Single
Dim PixelX, PixelY As Integer
Dim Npt(10) As POINTAPI
Dim Npt2(10) As POINTAPI
Dim OS As Integer
```

Dim HPrinterDPI：存放图片框与打印机间水平象素分辨率之差。

Dim VPrinterDPI：存放图片框与打印机间垂直象素分辨率之差。

Dim PixelX,PixelY：转换 twip 单位成象素单位(TwipPerPixel)。

Dim Npt(10) As POINTAPI：存放两个框对象中较小者的外围尺寸。

Dim Npt2(10) As POINTAPI：存放页边界框外围尺寸(页边)。

Dim OS：存放两个框对象中较小者的位置偏差值。

14.2　在图片框上绘制轮廓

按下标有 Draw1 的按钮则在 Art 图片框控制对象上绘制方框、圆和直线轮廓。在清单 14.2 的 Draw1_Click 事件中 Npt(index)值表示打印机页边。

清单 14.2　指定四个对象的绘制坐标

```
Sub Draw1_Click ()
    Npt(1).X = 100 \ PixelX: Npt(1).Y = 75 \ PixelY
    Npt(2).X = 100 \ PixelX: Npt(2).Y = 5200 \ PixelY
    Npt(3).X = 4000 \ PixelX: Npt(3).Y = 5200 \ PixelY
    Npt(4).X = 4000 \ PixelX: Npt(4).Y = 75 \ PixelY
Device% = Art.hDC
OS = 50: HPrinterDPI = 0: VPrinterDPI = 0
DrawObjects Device%
    Manual_Print.Enabled = True
    Print_API.Enabled = True
End Sub
```

Npt(1).X=100\PixelX:指定方框和直线对象的边界框(空白区),并将 twip 值转换成象素单位。Form_Load 事件包括以下代码行:

```
PixelX=Screen.TwipsPerPixelX
PixelY=Screen.TwipsPerPixelY
```

Device%=Art.hDC:使传送设备名至 API 绘制函数变得容易,本程序既可绘制在 Art 图片框中,也可绘制在打印机上。

OS=50:HPrinterDPI=0:VPrinterDPI=0:设置默认绘图坐标值。

DrawObjects Device%:调用 DrawObjects 过程(见清单 14.3)并传送绘制目标设备名至过程,此处设备为 Art 图片框控制对象。

14.3 绘制对象

按下 PrintAPI 窗体中三个主要按钮的任一个,清单 14.3 的 DrawObjects 过程将绘制每个对象到设备,Device As Integer 参数等于 hDC 设备名(图片框或打印机)。

清单 14.3 在设备上绘制四个对象

```
Sub DrawObjects (Device As Integer)
    API1 = Ellipse(Device, Npt(1).X, Npt(1).Y, Npt(3).X, Npt(3).Y)
    API2 = Polygon(Device, Npt(1), 4)
    API3 = Rectangle(Device, Npt(1).X + OS, Npt(1).Y + OS,
        Npt(3).X - OS, Npt(3).Y - OS)
    Npt2(1).X = Npt(1).X: Npt2(1).Y = Npt(1).Y
    Npt2(2).X = Npt(3).X: Npt2(2).Y = Npt(3).Y
    API4 = Polyline(Device, Npt2(1), 2)
End Sub
```

API1=Ellipse:绘制与页边相接的椭圆。

API2=Polygon:在页边周围绘制一个方框。

API3=Rectangle:在页中绘制一个较小矩形,OS 值等于从页边开始的偏移。

API4=Polyline:绘制一对角线。

14.4 打印机上页面的人工定标

清单 14.4 的 Manual_Print_Click 事件利用预置 HPrinterDPI 与 VPrinterDPI 值调整图片框与打印机之间的定标差,本例程也正是 Print_API_Click 事件使用的(参见清单 14.5)。APIH 与 APIV 值等于 API 函数 GetDeviceCaps 计算值,如清单 14.5 所示。

清单 14.4 测试打印机定标值

```
Sub Manual_print_Click ()
    Screen.MousePointer = 11
    Form1.Caption = "Printing page, please wait."
        Printer.PSet (0, 0) 'start printer
        LeftM = 0: TopM = 0 'adjust Margins

        APIH = Printer.ScaleWidth / Printer.TwipsPerPixelX
```

```
            APIV = Printer.ScaleHeight / Printer.TwipsPerPixelY
            HPrinterDPI = APIH / (Art.Width / Screen.TwipsPerPixelX)
            VPrinterDPI = APIV / (Art.Height / Screen.TwipsPerPixelX)
        For Z = 1 To 4
            Npt(Z).X = (Npt(Z).X * HPrinterDPI) - LeftM
            Npt(Z).Y = (Npt(Z).Y * VPrinterDPI) - TopM
        Next Z
    Device% = Printer.hDC
    OS = 50 * HPrinterDPI
    DrawObjects Device%
        Printer.EndDoc : Art.Cls
        Manual_Print.Enabled = False: Print_API.Enabled = False:
        Form1.Caption = "test print API pattern"
        Screen.MousePointer = 0
    End Sub
```

Screen. MousePointer＝11：改变鼠标指针成沙漏形状。

Printer. PSet(0,0)：有时会发现 API 图形函数并不立即在 Visual Basic 控制对象或打印页上开始绘制，为改变这种状况，需为欲绘制目标设备获取一个设备环境控制对象句柄，当使用 Basic 语言的图形、字体或打印语句时，Visual Basic 自动完成以上工作。通过在控制对象上放置简单象素点，为控制对象获取一个设备环境参考值，使用(0,0)坐标是因为在该位置永远看不见该象素。

LeftM＝0 ：TopM＝0：若图象不居中可调整页边。

APIH：APIV：通过打印机 ScaleWidth 与 ScaleHeight 除以每象素 twip 数，可获知反映打印尺寸的每英寸逻辑象素数。这与使用 API 函数 GetDeviceCaps 查询打印机 LogPixelSX(8)与 LogPixelSY(10)一致。

HPrinterDPI：可改变该值以使图象水平伸缩。对某些高 DPI 打印机，可能需要把此值乘上一个数(例如，对 600 dpi 打印机 HPrinterDPI * 2)。

VPrinterDPI：可改变该值以使图象垂直伸缩。对某些高 DPI 打印机，可能需要把此值乘上一个数(例如对 600 dpi 打印机 VPrinterDPI * 2)。

Npt(Z). X＝(Npt(Z) * HPrinterDPI)－LeftM：每个对象的 X 坐标。

Npt(Z). Y＝(Npt(Z). Y * VPrinterDPI)－TopM：每个对象的 Y 坐标。

Device%＝Printer. hDC：指定绘制目标设备名。

OS＝50 * HPrinterDPI：绘制两个方框对象中较小方框的页边偏移值。

DrawObjects Device%：调用 DrawObjects 过程(清单 14.3)，绘制对象到设备上(图片框或打印页)。

Printer. EndDoc：告知打印机打印该页。

Screen. MousePointer＝0：重置鼠标为箭头。

14.5 API 打印

按下标有 PrintAPI 的按钮激活清单 14.5 中的 Print_API_Click 事件，可使用 API 函数 GetDeviceCaps 获取关于设备能力的信息。为在打印机上打印显示器上显示的图象，必须计算两设备间分辨率之差。也可增加附加 API 打印设备调用以获取有关特定打印机的任何信

息，本清单包含可能获取或发送打印机信息的 API 函数，可参见 Windows SDK 参考手册。

清单 14.5 打印时使用 API GetDeviceCaps

```
Sub Print_API_Click ()
    Screen. MousePointer = 11
    Form1. Caption = "Printing page, please wait."
Printer. PSet (0, 0) 'start printer
APIH = GetDeviceCaps(Printer. hDC, 8)
APIV = GetDeviceCaps(Printer. hDC, 10)
HPrinterDPI = APIH / (Art. Width \ PixelX)
VPrinterDPI = APIV / (Art. Height \ PixelY)
LeftM = 0: TopM = 0 'adjust Margins
    For Z = 1 To 4
        Npt(Z). X = (Npt(Z). X - LeftM) * HPrinterDPI
        Npt(Z). Y = (Npt(Z). Y - TopM) * VPrinterDPI
    Next Z
Device% = Printer. hDC
OS = 50 * HPrinterDPI
DrawObjects Device%
    Printer. EndDoc : Art. Cls
    Print_API. Enabled = False: Manual_Print. Enabled = False
    Form1. Caption = "test print API pattern"
    Screen. MousePointer = 0
End Sub
```

Screen. MousePointer= 11：改变鼠标指针成沙漏形状。

Printer. PSet(0,0)：发送一图形语句以引起打印机注意。

APIH=GetDeviceCaps(Printer. hDC,8)：获取打印机水平分辨率。

APIV=GetDeviceCaps(Printer. hDC,10)：获取打印机垂直分辨率。

HPrinterDPI=APIH/(Art. Width\PixelX)：调整图片框与打印机之间的水平定标差，从该值加或减以水平定标页图象。

VPrinterDPI=APIV/(Art. Height\PixelY)：调整图片框与打印机之间的垂直定标差，从该值加或减以垂直定标页图象。

LeftM=0:TopM=0：若图象不居中则调整页边。

Npt(Z). X=(Npt(Z). X-LeftM) * HPrinterDPI：每个对象的 X 坐标。

Npt(Z). Y=(Npt(Z). Y-TopM) * VPrinterDPI：每个对象的 Y 坐标。

Device%=Printer. hDC：指定绘制对象使用的设备名(打印机)。

OS=50 * HPrinterDPI：两个框对象中较小者从页边开始的偏移。

DrawObjects Device%：调用 DrawObjects 过程(清单 14.3)，该过程绘制对象至设备(图片框或打印页)。

Printer. EndDoc：告知打印机打印该页。

Screen. MousePointer=0：重置鼠标为箭头。

14.6 PrintAPI 程序小结

使用 PrintAPI 程序给人一种如何打印 API 图形图象至打印页的普遍做法，尽管必须加

入更多的过程才能决定目标打印机功能，但这种小小的牺牲总比使用 Visual Basic 的 PrintForm 语句打印图象的屏幕硬拷贝要好得多。

重新规划 Art 图片框尺寸然后打印图象，图片框内部任何可见图象都被拉伸或缩小以适应打印页大小。在使用 twip 定标单位时，可以用 Art 图片框的 ScaleWidth 属性取代 HPrinterDPI 与 VPrinterDPI 公式中的 Art. Width 值，这可以更灵活地测试打印图象定标效果。

14.7　定标测试程序

测试 Visual Basic 图片框控制对象内部定标差公式时使用定标测试程序（Scaltest Program），以下部分包括调整尺寸缩小与尺寸放大工具性能的两个例子，Zoom_Basic 菜单使用 Visual Basic 定标语句，而 Zoom_API 菜单使用 API 函数，混合使用这两种方法容易产生灾难性后果，因而不建议在应用程序中这样做。Zoom_Basic 菜单标记的缩放百分比是虚值，并不表示真正的定标百分比。

拖放标尺标线至不同位置时，务必记住检查它们是否与鼠标对齐。在 Zoom_API 定标模式中标线并不对齐，这是由于 Ruler 事件未使用 API 函数 DLtoLP 转换鼠标坐标。使用 API 定标函数时亦需注意在标题控制对象上显示的鼠标当前 X 和 Y 值。

可在参考磁盘 14 SCALE 目录处找到 RULER. MAK 程序，Ruler 程序基本上是定标测试程序的拷贝，只是在每个程序中绘制标尺时使用的公式不同。使用水平或垂直滚动条时 Artisan，ArtAPI 和 Scaltest 程序中包含标尺的两个图片框将被移去，而在 Ruler 程序中，两个标尺保持不动。

另外还有一个较小的定标程序叫 ScaleAPI，它放在参考磁盘 16MISC 目录中，演示窗体或图片框中选定区域的伸缩。

注意：定标测试程序使用 Visual Basic 定标属性法和 API 定标函数，不要混用这两种方法，这样做只会导致定标例程执行期间一种方法与另一种方法的竞争，Visual Basic 窗体和图片框的定标属性使用一种与 API 对应部分完全不同的映射方法。

图 14.3 示出了 Scaletest 窗体及其控制对象。

定标测试程序窗体及其控制对象的属性列于表 14.2。

表 14.2　定标测试窗体及其控制对象的属性

Object：Form
Object ZoomTest

BackColor	=&H00FFFFFF&	ScaleMode	=3 'Pixel
Caption	="Zoom Scaling Test"	Top	=1125
Height	=5925	Width	=7245
Left	=105		

Object：Picture Box
Object StatusArea

BackColor	=&H00FF0000&	ScaleMode	=3 'Pixel
Height	=750	Top	=4440
Left	=60	Width	=7035

Object：HScroll Bar

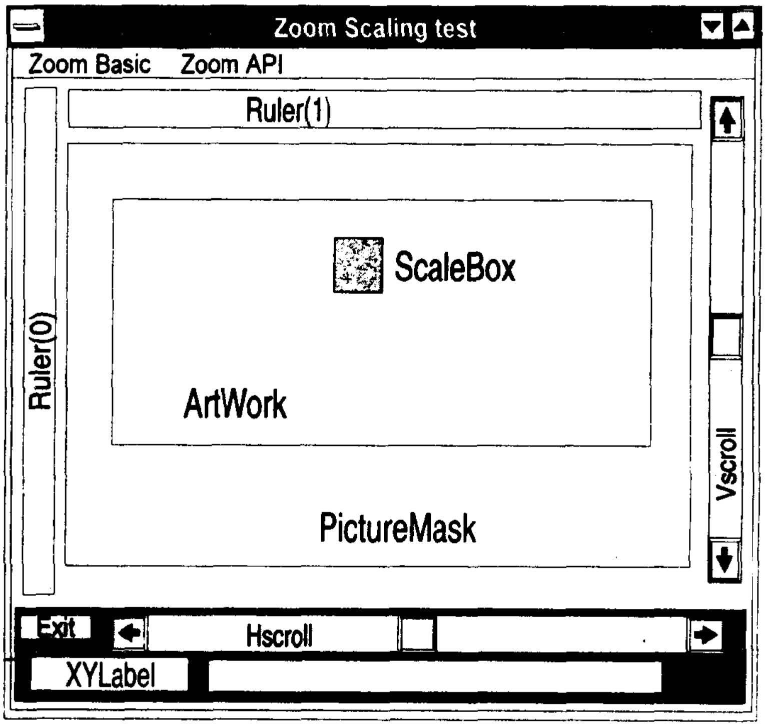

Zoom Scaling test
Zoom Basic
Zoom API
Ruler(1)
Ruler(0)
ScaleBox
ArtWork
PictureMask
Vscroll
Exit
Hscroll
XYLabel

图 14.3 定标测试窗体图例

Object HScrollBar

Height =285
LargeChange =100
Left =810
Max =200
Min =−200
SmallChange =10
Top =0
Width =6135

Object: Command Button

Object ExitBtn

Caption ="Exit"
Height =270
Left =90
Top =0
Width =645

Object: Label

Object StatusLabel

AutoSize =−1 'True
BackColor =&H00FFFFFF&
BorderStyle =1 'Fixed Single
ForeColor =&H00FFFFFF&
Height =285
Left =1800
Top =360
Width =4485

Object: Label

Object XYLabel

Alignment =2 'Center
BorderStyle =1 'Fixed Single
Height =285
Left =180
Top =405
Width =1500

Object: VScroll Bar

Object VScroll

Height =4245
LargeChange =100
Left =6720
Max =200
Min =−200
SmallChange =10
Top =0
Width =285

```
Object：Picture Box
Object Ruler
     BackColor         =&H00C0C0C0&          ScaleHeight        =300
     DrawMode          =7' Xor Pen           ScaleWidth         =6060
     Height            =330                  Top                =90
     Index             =1                    Left               =540
     Width             =6090

Object：Picture Box
Object Ruler
     BackColor         =&H00C0C0C0&          Left               =120
     DrawMode          =7' Xor Pen           Top                =120
     Height            =4245                 Width              =330
     Index             =0

Object：Picture Box
Object PictureMask
     BorderStyle       =0 'None              ScaleMode          =3 'Pixel
     Height            =3480                 Top                =540
     Left              =540                  Width              =6000

Object：Picture Box
Object ArtWork
     BackColor         =&H0000FFFF&          Left               =420
     BorderSytle       =0 'None              Top                =660
     Height            =1770                 Width              =5235

Object：Picture Box
Object ScaleBox
     BackColor         =&H000000FF&          ScaleWidth         =300
     Height            =330                  Top                =270
     Left              =2565                 Width              =330
     ScaleHeight       =300

Menu ZoomBasic
     Caption           ="Zoom &Basic"

Menu Z_Basicsub
     Caption           ="25%"
     Index             =0

Menu Z_Basicsub
     Caption           ="50%"
     Index             =1

Menu Z_Basicsub
     Caption           ="75%"
     Index             =2

Menu Z_Basicsub
     Caption           ="100%"               Index              =3

Menu Z_Basicsub
     Caption           ="125%"               Index              =4

Menu Z_Basicsub
     Caption           ="150%"               Index              =5

Menu Z_Basicsub
     Caption           ="175%"               Index              =6

Menu Z_Basicsub
```

```
    Caption          ="200%"          Index          =7
Menu Z_Basicsub
    Caption          ="Full Page%"    Index          =8
Menu ZoomAPI
    Caption          ="Zoom &API"
Menu Z_APIsub
    Caption          ="2000"          Index          =0
Menu Z_APIsub
    Caption          ="4000"          Index          =1
Menu Z_APIsub
    Caption          ="-6000"         Index          =2
Menu Z_APIsub
    Caption          ="Full Page"     Index          =3
```

图 14.4 示出了字标测试程序的一个屏幕。

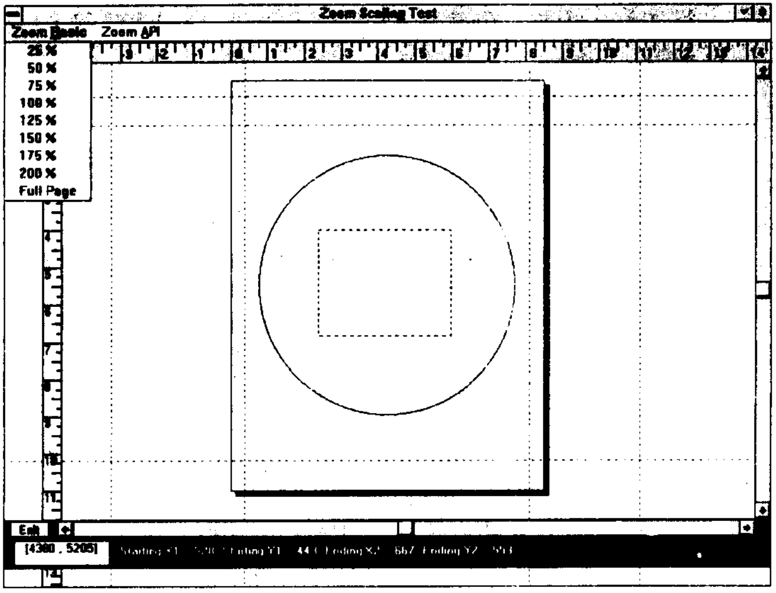

Zoom Scaling Test
Zoom Basic
Zoom API
25%
50%
75%
100%
125%
150%
175%
200%
Full Page
Exit
[4300, 5205]

图 14.4　定标测试程序的一个屏幕显示

14.8　声明与初始值

清单 14.6 声明部分使用了与第 2 章 Artisan 程序一致的变量名和值。

清单 14.6　初始变量名

```
Declaration:
Dim WIDE, HIGH As Integer
Dim PixelX, PixelY As Integer
Dim BSM, SSM As Single
Dim StartPt As Integer
Dim ASL, AST, ASW, ASH As Integer
Dim Ruler0X, Ruler0Y As Long
Dim Ruler1X, Ruler1Y As Long
Dim X1, Y1, X2, Y2 As Long
Dim XX1,YY1,XX2,YY2 As Long
Dim OldSW As Long
Dim OldSH As Long
Dim OldRH As Long
Dim OldRW As Long
Dim PerSW As Single
Dim PerSH As Single
Dim XY(1 To 4) As POINTAPI
```

WIDE and HIGH：切换显示器分辨率，VGA 与 SVGA 通常是每英寸 96 象素。

PixelX and PixelY：表示 Visual Basic 参考语言手册定义的 TwipsPerPixelX 和 TwipsPerPixelY 值。

BBM（底滚动移动）和 SSM（边滚动移动）：移动 Artwork 图片框和 Rulers 图片框时滚动条使用的偏移。

StartPt：在屏幕上绘制选取框（marquee box）时初始化坐标。

Dim ASL，AST，ASW，ASH As Integer：存放 Artwork 的 ScaleLeft，ScaleTop，ScaleWidth 和 ScaleHeight 值。

Ruler0X，Ruler0Y，Ruler1X，Ruler1Y：绘制标尺刻度线时临时直线坐标，它们分别为标尺 Line 语句使用 X1，Y1，X2 及 Y2 值。

OldSW，OldSH，OldRH，OldRW：存放 Arwork 与 Ruler 图片框 ScaleLeft 和 ScaleHeight 旧值。

PerSW，PerSH：缩放后重新定标控制对象使用的手工偏移。

14.9 图片框鼠标事件

Artwork 图片框鼠标事件简单地在桌面上绘制点线组成的框轮廓。定标测试程序演示两种根本不同的定标方法（API 定标和 Visual Basic 定标），由于 API 函数要求象素单位的绘制坐标，Artwork 桌面则使用用户自定义定标，因而必须转换坐标值，获取设备逻辑点坐标的一种方法是使用 API 函数 DPtoLP，如清单 14.8 所示。

清单 14.7 中的 Artwork_MouseDown 事件保存 Artwork 图片框当前定标数组，任何以象素为单位的图和任何定标图象的重绘都是用户定义的单位为基准的。

清单 14.7　设置图片框定标模式

```
Sub ArtWork_MouseDown (Button%, Shift%, X%, Y%)
    ASL = ArtWork.ScaleLeft: AST = ArtWork.ScaleTop
    ASW = ArtWork.ScaleWidth: ASH = ArtWork.ScaleHeight
    ArtWork.DrawMode = 6: ArtWork.DrawStyle = 2
    ArtWork.ScaleMode = 3
    StartPt = True
End Sub
```

ASL＝ArtWork.ScaleLeft：保存图片框所有当前定标数组。

ArtWork. DrawMode：改变绘制模式为反转(Inverse(6))。

ArtWork. DrawStyle：改变绘制风格为点型(**Dots**(2))，以绘制虚线外形框。

ArtWork. ScaleMode：改变绘制定标模式为象素(**Pixel**(3))模式。

StartPt = True：除了初始化 MouseDown 框中起始点外，还使用 StartPt 变量作为 MouseMove 事件设置坐标的标志。

清单 14.8 中的 ArtWork_MouseMove 事件在图片框中绘制虚线矩形，由于程序使用两种定标方法(API 定标和 Visual Basic 定标)，API 函数 DPtoLP 转换 X 和 Y 坐标以取得正确屏幕坐标。

清单 14.8 使用设备点至逻辑点转换

```
Sub Artwork_MouseMove (Button%, Shift%, X!, Y!)
    Xstr = Format $ (X, "000"): Ystr = Format $ (Y, "000")
    XYLabel.Caption = "[" + Xstr + " , " + Ystr + "]"
RulerGuides Index%, Button, Shift, X, Y
    If SideRuler = True Or TopRuler = True Then
        GuideLines Index%, Button, X, Y
    End If
    If Button Then
        XY(1).X = X
        XY(1).Y = Y
        API = DPtoLP(ArtWork.hDC, XY(1), 1)
        XX = XY(1).X
        YY = XY(1).Y
If StartPt = True Then StartPt = False: W = XX: N = YY: E = XX: S = YY
  API = Rectangle(ArtWork.hDC, W, N, E, S)
    E = XX: S = YY
  API = Rectangle(ArtWork.hDC, W, N, E, S)
  Msg $ = "Starting X1 = " + Format $ (W) + " "
  Msg $ = Msg $ + "Starting Y1 = " + Format $ (N) + " "
  Msg $ = Msg $ + "Ending X2 = " + Format $ (E) + " "
  Msg $ = Msg $ + "Ending Y2 = " + Format $ (S) + " "
  StatusLabel.Caption = Msg $
End If
End Sub
```

XY(1). X=X：把当前鼠标指针的 X 坐标存放在 Npt 数组中。

XY(1). Y=Y：把当前鼠标指针的 Y 坐标存放在 Npt 数组中。

API=DPtoLP(ArtWork. hDC,XY(1),1)：API 函数 DPtoLP 转换设备点为逻辑点，此时视设备点为 Artwork 图片框控制对象，使用从控制对象左上角开始的象素度量坐标系统。逻辑点是定标图片框时任何新的 API 内部定标数组。在使用 Visual Basic 的 ScaleWidth 与 ScaleHeight 手工改变控制对象内部定标时，设备点与逻辑点自动完成转换，使用 API 定标函数改变内部定标时，控制对象的 ScaleWidth 与 ScaleHeight 并不改动，使用 DPtoLP 函数手工设置成合适的值，这种能力仿真 Visual Basic 定标宽度与高度语句。

XX=XY(1).X：存放新 X 坐标。

YY=XY(1).Y：存放新 Y 坐标。

If StartPt=True Then：初始化 MouseMove 事件而不是 MouseDown 事件的框的起始点与终止点，由于参数 X As Single 和 Y As Single 在 API 函数 DPtoLP 转换它们之前使用象素单位，所以在本事件中进行初始化。

StartPt+False：MouseMove 期间框中起始点只需初始化一次。

W=XX:N=YY：指定框的起点与终点坐标。

API=Rectangle：利用 API 函数绘制框轮廓。

E=XX:S=YY：建立与当前鼠标位置一致的边界框角位置。

清单 14.9 中的 Artwork_MouseUp 事件重置图片框最初的 Visual Basic 定标数组。

清单 14.9　恢复桌面定标

```
Sub ArtWork_MouseUp (Button%, Shift%, X!, Y!)
    ArtWork.DrawMode = 6: ArtWork.DrawStyle = 0
    ArtWork.ScaleLeft = ASL: ArtWork.ScaleTop = AST
    ArtWork.ScaleWidth = ASW: ArtWork.ScaleHeight = ASH
End Sub
```

ArtWork.ScaleLeft：重置定标使标尺正常工作，若使用 Visual Basic Scale(1,2)-(3,4)语句，本语句不能正确执行，必须单独地为每个定标属性设置值。

14.10　初始化定标值对齐

ScaleBox 图片框在 Artwork 图片框控制对象内部，当 Artwork 内部定标改变时，ScaleBox 自动放置至新的内部位置，Ruler_Paint 事件用 ScaleBox 新位置作为正确内部定标准则和标尺。

清单 14.10 的 Form_Activate 事件保存 Artwork 图片框原始定标宽度与定标高度，以便运行 API 定标函数后重置。

清单 14.10　开始时保存定标值

```
Sub Form_Activate ()
    ScaleBox.Top = 3750
    ScaleBox.Left = 3600
    ScaleBox.Width = 330
    ScaleBox.Height = 330
    OldSW = ArtWork.ScaleWidth: OldSH = ArtWork.ScaleHeight
    OldRW = 300
    OldRH = 300
        Ruler(0).Scale: Ruler(1).Scale
End Sub
```

ScaleBox.Top=3750：ScaleBox 是一个 Visible 属性为 False 的图片框控制对象，可用 ScaleBox 在桌面上的预定位置正确地对齐标尺刻度线，由于标尺是分散图片框，只能在

每个标尺内部定标反映 Artwork 图片标内部定标的正确值，ScaleBox 直接定位在两标尺交点下面。开始时，控制对象的 Left 属性与 Ruler(0)的 Scaleleft 属性初始值相等，Top 属性也与 Ruler(1)的 ScaleTop 属性初始值相等。

OldSW＝ArtWork. ScaleWidth：存放缩放前的定标宽度值。

OldSH＝ArtWork. ScaleHeight：存放缩放前的定标高度值。

OldRW＝300：存放缩放前 Ruler(0)的定标宽度值。

OblRH＝300：存放缩放前 Ruler(1)的定标高度值。

14.11 Zoom API 菜单

在 Zoom API 菜单项中选择一个子菜单项激活 Z_APIsub_Click 事件，清单 14.11 的 Zoom API 例程用 API 定标函数 ViewPorts 仿真 Visual Basic 标定法，使用 API 定标函数的优越性在于它们对用 API 图形函数绘制图象的重新定标能力，例如，一个只有一个象素宽的对象被 API 定标函数放大时，框线自动地增加以反映新的视窗(Viewport)定标。

清单 14.11 利用 API 函数放大

```
Sub Z_APIsub_Click (Index As Integer)
ArtWork. Scale 'reset original size
   Select Case Index
     Case 0
        Area = 2000
     Case 1
        Area = 4000
     Case 2
        Area = -6000
     Case 3

          HScroll. Value = 0: VScroll. Value = 0
          PerSW = 1: PerSH = 1
          UpdateObjects
          Exit SubEnd Select
ArtWorkScale% = SaveDC%(ArtWork. hDC)
     SW = ArtWork. ScaleWidth
     SH = ArtWork. ScaleHeight
API = SetMapMode(ArtWork. hDC, 8)
API = SetWindowOrg(ArtWork. hDC, 0, 0)
API = SetWindowExt(ArtWork. hDC, SW, SH)
API = SetViewportExt(ArtWork. hDC, SW - Area, SH - Area)
API = SetViewportOrg(ArtWork. hDC, (Area \ 2) \ PixelX, (Area \ 2) \ PixelY)
          UpdateObjects
API = RestoreDC%(ArtWork. hDC, ArtWorkScale%)
End Sub
```

ArtWork. Scale：为了使本程序使用的两种定标方法通用起见，重置定标为 twip。

Select Case Index：根据所选择的菜单项，完成相应的 Case 模块。

Case 0,1,2,Area＝2000,4000,－6000：指定 Artwork 图片框视窗宽度。

Case 3：恢复原先的初始定标值和页尺寸。

HScroll. Value＝0：VScroll. Value＝0：重置滚动条至中间位置。

PerSW＝1：PerSH＝1：重置滚动条移动百分比值。

UpdateObjects：重绘整页至桌面。

ArtWorkScale%＝SaveDC%（ArtWork. hDC）：保存 Artwork 图片框前一设备环境。

API＝SetMapMode（ArtWork. hDC，8）：设置所用的定标模式，MM_ANISOTROPIC 值是 8，这与 Visual Basic 的 ScaleMode 为 0 相似。

API＝SetWindowOrg（ArtWork. hDC，0，0）：图片框控制对象的前 ScaleLeft 和 ScaleTop 值。

API＝SetWindowExt（ArtWork. hDC，SW，SH）：图片框控制对象的前 ScaleWidth 和 ScaleHeight 值。

API＝SetViewportExt：新 ScaleWidth，ScaleHeight 值。

API＝SetViewportOrg：新 ScaleLeft，ScaleTop 值。

UpdateObjects：重绘桌面内容至新 API 定标空间。

14.12　Zoom Basic 菜单

在 Zoom Basic 菜单项中选择一个子菜单项将触发 Z_Basicsub_Click 事件，清单 14.12 的 Zoom Basic 例程用标准 Visual Basic 图片框定标属性缩小或放大桌面图象，图片框 ScaleLeft，ScaleTop，ScaleWidth 与 ScaleHeight 新属性在此初始化，实际定标变动在 Zoom 过程中完成（参见清单 14.13）。

清单 14.12　用 Visual Basic 语句放大

```
Sub Z_Basicsub_Click (Index As Integer)
ArtWork.Scale 'restore scale or scaling will be infinite
''''''Index, ZLeft, ZTop, ZWidth, ZHeight
  Select Case Index
     Case 0 '25
          Zoom 2, -5000, -5000, 10000, 8500
     Case 1 '50
          Zoom 2, -4000, -4000, 8000, 6800
     Case 2 '75
          Zoom 2, -3000, -3000, 6000, 5100
     Case 3 '100
          Zoom 2, -2000, -2000, 4000, 3400
     Case 4 '125
          Zoom 2, 2000, 2000, -4000, -3400
     Case 5 '150
          Zoom 2, 3000, 3000, -6000, -5100
     Case 6 '175
          Zoom 2, 3750, 3750, -7500, -6375
     Case 7 '200
          Zoom 2, 5000, 5000, -10000, -8500
     Case 8 'Full Page
          ArtWork.Scale
          HScroll.Value = 0: VScroll.Value = 0
          PerSW = 1: PerSH = 1
```

```
            UpdateObjects
            Ruler(0).ScaleTop = ArtWork.ScaleTop
            Ruler(0).ScaleHeight = ArtWork.ScaleHeight
            Ruler(1).ScaleLeft = ArtWork.ScaleLeft
            Ruler(1).ScaleWidth = ArtWork.ScaleWidth
    End Select
    Ruler(0).Refresh
    End Sub
```

Select Case Index：根据选择的菜单项，完成与其序号值相匹配的Case 模块。

Case 0 to 7：利用 Visual Basic 定标值给图片框控制对象重定标，改变定标值的过程名为 Zoom(清单 14.13)，传送的参数有：

1. 选择菜单的序号。
2. ScaleLeft 属性。
3. ScaleTop 属性。
4. ScaleWidth 属性。
5. ScaleHeight 属性。

Case 8：重置启动时使用的原始值。

ArtWork.Scale：重置定标模式为 Twip。

HScroll.Value=0:VScroll.Value=0：重置滚动条至中间位置。

PerSW=1:PerSH=1：重置滚动条移动百分比值。

UpdateObjects：重绘桌面页内容。

Ruler(0).ScaleTop,Height,Left,Width：重置标尺内部定标。

Ruler(0).Refresh：画标尺显示 inch-mark 图象。

14.13 重定标图片框控制对象

清单 14.13 的 Zoom 过程用清单 14.12 的 Z_Basicsub_Click 事件传送的参数重定标图片框，此法即简单地用图片框当前定标值加上或减去 Z_Basicsub_Click 事件传来的定标值，保存新的定标数组，两个标尺图片框定标值亦作调整。

清单 14.13 用 Visual Basic 语句重定标桌面

```
Sub Zoom (Index%, ZLeft%, ZTop%, ZWidth%, ZHeight%)
    ArtWork.ScaleLeft = ArtWork.ScaleLeft + ZLeft
    ArtWork.ScaleTop = ArtWork.ScaleTop + ZTop
    ArtWork.ScaleWidth = ArtWork.ScaleWidth + ZWidth
    ArtWork.ScaleHeight = ArtWork.ScaleHeight + ZHeight
NewSW = ArtWork.ScaleWidth: NewSH = ArtWork.ScaleHeight
PerSW = NewSW / OldSW: PerSH = NewSH / OldSH
    BSM = BSM * PerSW: SSM = SSM * PerSH
'''''''''''
        UpdateObjects
    Ruler(0).ScaleTop = ArtWork.ScaleTop
    Ruler(0).ScaleHeight = ArtWork.ScaleHeight
    Ruler(1).ScaleLeft = ArtWork.ScaleLeft
    Ruler(1).ScaleWidth = ArtWork.ScaleWidth
End Sub
```

ArtWork.ScaleLeft,Top,Width,Height：重定标 ArtWork 图片框控制对象内部数组。

NewSW.NewSH：保存 ScaleWidth 和 ScaleHeight 新值。

PerSW＝NewSW/OldSW：计算 ScaleWidth 新旧值之差。

PerSH＝NewSH/OldSH：计算 ScaleHeight 新旧值之差。

BSM＝BSM * PerSW：计算底部滚动条移动(Bottom ScrollBar MoveMent)与 PerSH 值的积，BSM 在滚动条 Change 事件中计算(HScroll.Value PixelX) * PerSW)，以便计算用户可能使用的桌面滚动偏移。

SSM＝SSM * PerSH：计算侧部卷滚条移动(Side Scrollbar Movement)。

UpdateObjects：重绘桌面页内容。

Ruler(0).ScaleTop,Height：重置标尺内部定标。

14.14　本章小结

Visual Basic 定标模式和 API 定标函数能在显示器的图形显示上产生不同的效果。API 定标函数通常告知 GUI 调整 API 绘制属性以反映当前定标数组，若绘制一个只有一个象素的边框，则调用 API SetViewportOrg 和 SetViewPortExt 放大它，框线自动增大，反映了新定标数组。但改变 Visual Basic 定标模式与 API 定标函数的连接时宽度不变，与绘制包 Wireframe 模式相似。

Visual Basic 定标性能使得改变内部定标变得简易，只要使用 Scale，ScaleLeft，ScaleTop，ScaleWidth 和 ScaleHeight 关键字即可，然而你会注意到宽度保持不变，精确地放置实际绘图是很容易的，但必须为每个绘制对象设计一个公式以适应可能产生的外形宽度变化。使用 API 定标函数的一个好处是它们自动完成这个工作，只是使用这些函数可能需要将设备点转换成逻辑点。

如果决定在所有控制对象中都使用象素定标模式，值得注意的是必须对 Rulers，Guidelines，Scalebox 和 Scrollbars 作修改以反映象素模式，同时记住手工绘制 Bézier 曲线时需要 Twip 或更细的定标模式，这样才能在使用 deCasteljau 除法公式时精确地将每个象素点定位到曲线点上。

第 15 章　ArtAPI 程序

ArtAPI 程序使用了一个类似于第 13 章“节点连接程序”中讨论过的例程。此例程连接线段与贝塞尔曲线以形成折线(Polyline)或多边形(polygon shape)。大部分过程是由第 2 章 Artisan 程序中的过程修改而成，因而易于理解。ArtAPI 窗体的外观与 Artisan 窗体的外观明显不同。在 Form_Resize 事件中，可找到新控制对象及其在窗体上的位置的清单。窗体外观的最大变化是添加了 PicClip. VBX 控制对象，该控制对象是用来把所有位图式工具图象放在一个图片框中。MenuBitmaps 过程(见 ARTAPI1. FRM 源代码)提取此图片框中每个位图并将每幅图象放入窗体菜单栏的指定子菜单中。

15.1　图柄与节点

为了增强 Artisan 程序的性能，原来的图柄和节点图片框被替换以 Image(图象)控制对象。图象控制对象没有内部缩放比例属性，因此所有缩放调节函数都从程序中去掉了。这样，缩放工具基本上没用，除非编写一个过程，当重新调节 Artwork 图片框桌面时，才可以重新定位所有可视的图柄及节点。但是这样做是浪费时间和不切实际的，特别是为一个复杂的对象形状使用了许多节点时。一个解决办法是使用 API 函数 Rect(如 SetRect，SetRectEmpty，CopyRect 及 PtInRect)来绘制和定位所有图柄和节点，然后用 Artwork 图片框控制对象的鼠标事件来控制节点与图柄的移动。

15.2　坐标系统与缩放工具

直接在桌面(图片框)上绘制节点和图柄是相当令人厌烦的，但是通过使用前面介绍过的方法，你在理解用 C 进行 Windows 编程的基本原理方面已经超前一步了。与其在缩小和放大时编写很长的重新调节过程来重定位每个图柄和节点，不如使用自定义的 Rect 结构，因为它是桌面内部尺寸的一部分。这样，当重新调节桌面的任何区域时，图柄、节点及其他图形图象会自动调节到新的视窗或标定尺寸处。如果不需要缩放特性或者程序中只用到有限的图形，仍然以可选用本书中讨论的手工例程。

15.3　桌面及打印机 API 图形

在 ArtAPI 程序中大部分 Basic 语言里用于桌面和打印机的图形语句均被替换为 API 图形函数。

Visual Basic 使用自己的标定例程来定位窗体或图片框中的 Line 和 Circle 语句。这样，重新调节图形就比使用设备对设备的刻度偏移更容易。

不过，在使用 API 图形函数时，你将面对更大的挑战：即如何正确地在一给定设备(图片框或打印机中定位坐标。因为 API 和 Visual Basic 控制对象的坐标属性不相同。

在第 2 章开头已介绍过，使用 API 函数 GetDeviceCaps 来正确显示某未知设备(如显示器或打印机)上的图象是很必要的。每个设备都可能有各种内部坐标系统及分辨率输出。

15.4　ArtAPI 程序中的图形 API 函数

ArtAPI 程序使用了许多 API 函数，下面列出了其中一部分：

PolyLine：替换 Line 语句来绘制一条或多条相接的直线。

Polygon：绘制并填充复杂的闭合形状。

Ellipse：替换 Circle 语句。椭圆或圆绘制在拉伸并定出大小的一个不可见的边界框中。如果希望在绘制时鼠标接触该图形的边界，必须自己移动鼠标。

Rectangle：替换带 B 或 BF 选项的 Line 语句。对所有 API 图形，你都要添加代码来设置 Pen（笔）或 Brush（刷子）的颜色。Visual Basic 可使用 ForeColor 和 FillStyle 属性来自动修改 Pen 和 Brush 属性。

15.5　ArtAPI 程序中的主要变动

在第 2 章中 Artisan 程序的 ArtAPI 版本的基础上新增 18 个新过程。另外，原来 Artisan 程序中的几个控制对象也移到了过程调用中以处理多重 Bézier 曲线。例如 BezNode1 和 BezNode2 控制对象及其鼠标事件。

本章没有重复 Artisan 程序中的过程和控制对象，而是列出了有较大改动的过程。一些事件根据新变量清单有一些较小的变动。你最好对照着察看一下第 2 章到第 5 章中的代码并与本章作一比较。

在本章中一些过程使用了较长的代码例程，这是因为它们使用了 Visual Basic Image（图象）控制对象来模仿节点和图柄的屏上（on-screen）定位。图 15.3 将帮助你理解用来载入、卸载及移动节点和图柄的过程。

15.6　新的图片框控制对象

几个新的控制对象和过程使用 API 图形语句来绘制多边形对象。其中包括 ArtMask 图片框和 Pnode 图象控制对象。

ArtMask：用来寻找屏幕上每个对象的图片框。在原来的 Artisan 程序中，通过气泡法搜索整个边界框数组，可以找到某对象。当找到某对象（正好位于鼠标处）时，该对象被高亮显示。当几个边界框（对象）互相重叠时，这个方法就行不通了。如果某对象的边界框同另外的对象重叠，那么将不可能拾取或移动该对象。

使用 ArtMask 图片框选取一对象，只要接触对象的轮廓即可。为达到此目的，使用了一个图片框来模仿一次屏外绘制，图片框的 AutoRedraw 属性设为 True，Visible 属性设为 False。

Pnode：一个图象控制对象，代表向一个复杂多边形对象中加入或从中移去所有节点。本程序中仍有原来的节点控制对象，Pnode 的作用与其类似。Pnode 仅放在多边形对象周围。所有 Pnode 的鼠标事件例程可以放入原节点控制对象的鼠标事件中，但这会导致大量的事件代码。

15.7 变为过程的原有控制

第 2 章中原来 Artisan 程序的几个控制对象在此变为过程。下面介绍其中一部分。

BezNode1,BezNode2(图片框):第 2 章 Artisan 程序中的这两个图片框,在 ArtAPI 程序中被删去了。BezNode1 和 BezNode2 的鼠标事件移到下列控制对象的事件中。

BezNode_Down:处理关于多重 Bézier 曲线的 MouseDown 事件。

BezNode_Move:处理关于多重 Bézier 曲线的 MouseMove 事件。

BezNode_UP:处理关于多重 Bézier 曲线的 MouseUp 事件。

15.8 其他改动

本章同时也修改了第 2 章中原 Artisan 程序中的如下几个过程:

Find_Object:本过程现用 API 函数 PtInRect 来找出某对象的边界框。Find_Object 过程处理某对象在 ArtMask 图象控制对象中的拷贝。

MenuBitmaps:将位图图片载入窗体的 Menu。

mnuFileOpen:加入四行代码以打开保存在文件中的多边形。

mnuFileSave:加入四行代码以保存多边形。

UpdateArtMask:同 UpdateObject 过程类似,不同的是 UpdateArtMask 过程在 ArtMask 图象控制中只绘制一个对象。

UpdateObjects:除了用 API 图形函数绘制对象外,同第 5 章中原 UpdateObject 过程类似。

WinPrint:使用人工调节方法在打印页上放置 API 图形。请调整 LeftM,RightM,TopM 及 BottomM 变量以适应打印机输出。

15.9 新的多边形过程

有 12 个新过程被用来操作折线或多边形图象。在所有过程中,节点和图柄是必须围着图象定位的图象控制对象。在一个复杂形状周围放置图象控制对象(节点和图柄)需要有大量的编码。为了检查当你移动一节点或拉伸一多边形时发生的情况,最好的方法是用 F8 键或 Visual Basic 的单步按钮。移动图象的一个多重曲线部分看起来简单,实际上最多要有 8 个不同的过程和事件调用。

所有多边形过程都是分段编码的,这样可使你更好地理解每个过程的用途。如果你有 Windows 的 32 位操作系统,则可除去前面清单中的几个过程。32 位 GDI 库包括使用较少代码创建 Bézier 曲线的 API 函数,还有几个新的 move object to (移动对象)函数。请参看第 8 章 APIdraw 程序以了解 32 位 API 函数名。

12 个新的多边形过程如下:

Poly_CurveMove1:在闭合多边形中移动曲线段。

Poly_CurveMove 2:在开多边形中移动曲线段。

Poly_Curves:向屏幕、ArtMask 或打印机绘制 Bézier 曲线所用的三个独立的定标例程。

Poly_gon:形成一条多边形线段。

Poly_Handles：在多边形周围放置拉伸图柄。

Poly_LineMove1：在闭合多边形中移动线段。

Poly_LineMove2：在开多边形中移动线段。

Poly_Lines：如果用户将焦点从绘图切换到其他处，则绘制折线。

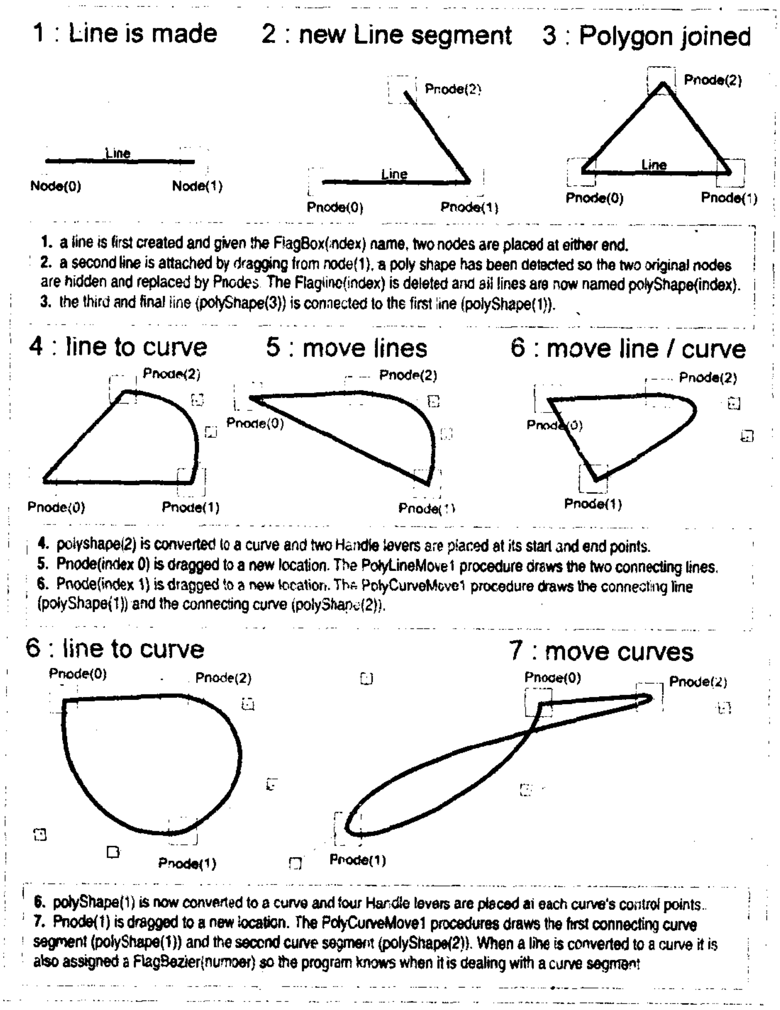
Poly_Nodes：在为折线添加线段时载入新节点。

Poly_Reload：当选取一多边形时，重新加载正确的节点数目。

Poly_Stretch：拉伸或翻转多边形。

Poly_Update：重置节点的可视属性。

前面这些多边形例程处理多边形的能力是有限的。需要另写代码来打断或删去节点连接，同时一次只能移动一个节点。折线或多边形只能按连续路径绘制，你不能通过改变始末点坐标来延伸线段。多重 Bézier 曲线限于尖端连接而不再具有平滑或对称连接的特点。图 15.1 显示了多边形的构成。


1 : Line is made
2 : new Line segment
3 : Polygon joined
Line
Node(0)
Node(1)
Pnode(2)
Line
Pnode(0)
Pnode(1)
Pnode(2)
Line
Pnode(0)
Pnode(1)
4 : line to curve
5 : move lines
6 : move line / curve
Pnode(2)
Pnode(0)
Pnode(1)
Pnode(2)
Pnode(0)
Pnode(1)
Pnode(2)
Pnode(0)
Pnode(1)
6 : line to curve
7 : move curves
Pnode(0)
Pnode(2)
Pnode(1)
Pnode(0)
Pnode(2)
Pnode(1)

1. a line is first created and given the FlagBox(index) name, two nodes are placed at either end.
2. a second line is attached by dragging from node(1), a poly shape has been detected so the two original nodes are hidden and replaced by Pnodes. The Flagline(index) is deleted and all lines are now named polyShape(index).
3. the third and final line (polyShape(3)) is connected to the first line (polyShape(1)).

4. polyshape(2) is converted to a curve and two Handle levers are placed at its start and end points.
5. Pnode(index 0) is dragged to a new location. The PolyLineMove1 procedure draws the two connecting lines.
6. Pnode(index 1) is dragged to a new location. The PolyCurveMove1 procedure draws the connecting line (polyShape(1)) and the connecting curve (polyShape(2)).

6. polyShape(1) is now converted to a curve and four Handle levers are placed at each curve's control points.
7. Pnode(1) is dragged to a new location. The PolyCurveMove1 procedures draws the first connecting curve segment (polyShape(1)) and the second curve segment (polyShape(2)). When a line is converted to a curve it is also assigned a FlagBezier(number) so the program knows when it is dealing with a curve segment

图 15.1　多边形的构成

15.10 To_Front_Back 例程

To_Front_Back 把一个对象移动到其他对象的前面或后面。原来的 Artisan 程序通过修改每个对象的次序和相关的属性来做到这一点。现在只需用一个既短小又简单的循环例程即可改变在桌面上绘制对象的次序。所要求的只是加上一个数组 DrawOrder，来保持并记录绘图的次序。UpdateObjects 根据记录在 DrawOrder 数组中的次序，用一个 For/Next 语句绘制每个对象。

15.11 API 定标系统和 WinPrint（窗口打印）

本程序使用了三个不同的定标例程来向 Artwork 图片框、ArtMask 图片框及打印机绘制 API 图形。API 图形函数在某设备（Visual Basic 控制或打印机页面）上绘图时，把左上角作为（0，0）起始坐标。使用 Visual Basic 的 Scale 属性来修改这些设置将不会影响坐标。API 库包含有几个函数来修改内部坐标系统或转换逻辑和设备坐标。ArtAPI 程序仍然用手工技术来把每个对象定位于图片框或打印机页面。

15.12 测试（Test）按钮

Test（测试）按钮包含一小段例程，它使用 Visual Basic 的 Line 语句重绘一个多边形。实际工作时，它将每条线段置以不同的颜色，曲线段的绘制使用 BezDraw 过程。

15.13 缩放工具

本程序没有连接缩放工具。因为使用了 Visual Basic 的图象控制对象作为节点，如果要使用缩放工具，就需要大量的手工调节和放置例程。可以参考第 14 章来找到在程序中集成缩放工具的方法。

ArtAPI 程序的窗体和控制对象属性列于表 15.1 中。

表 15.1 ArtAPI 窗体及其控制对象的属性

Object: Form
Object Name: ArtAPI

BackColor	=&H00404040&	Caption	="Mini API Graphic Shell"
ClipControls	=0 'False	ForeColor	=&H00FFFFFF&
Height	=6735	Left	=60
ScaleMode	=3 'Pixel	Top	=1065
Width	=7110	WindowState	=2 'Maximized

Object: PictureBox
Object Name: ToolButtons

Align	=1 'Align Top	AutoSize	=-1 'True
BackColor	=&H00C0C0C0&	BorderStyle	=0 'None
ClipControls	=0 'False	DrawMode	=6 'Invert
Height	=540	Left	=0
Picture	=ToolButton. BMP	ScaleMode	=3'Pixel
Top	=0	Visible	=0'False

Object: ComboBox

```
ObjectName: Combo1
    BackColor      =&H00FFFFFF&        Height        =300
    Left           =4380               Style         =2 'Dropdown List
    TabIndex       =20                 Top           =240' within ToolButtons
    Width          1815

Object:PictureBox
Object Name: StatusColor
    Height         =450                Left          =6300
    ScaleMode      =3 'Pixel           Top           =60' within ToolButtons
    Visible        =0 'False           Width         =630

Object:PictureBox
Object Name: StatusPaint
    Height         =315                Left          =120
    Top            =60 'within StatusColor  Visible  =0' False
    Width          =345

Object:Label
Object Name: StatusLabel
    Alignment      =2 'Center          BackColor     =&H00FFFF00&
    BorderStyle    =1 'Fixed Single    Height        =255
    Left           =4380               Top           =0 'within ToolButtons
    Visible        =0' False           Width         =1575

Object: CommonDialog
Object Name:CMDialog1
    CancelError    =-1 'True           Left          =0
    Top            =4320

Object: PictureBox
Object Name: StatusArea
    Align          =2 'Align Bottom    BackColor     =&H00000000&
    BorderStyle    =0 'None            ForeColor     =&H00FFFFFF&
    Height         =915                Left          =0
    ScaleMode      =3 'Pixel           Top           =5160
    Visible        =0 'False           Width         =7020

Object: PictureBox
Object Name: ColorMask
    ClipControls   =0 'False           Height        =435
    Left           =45                 ScaleMode     =3 'Pixel
    Top            =405 'within StatusArea  Visible  =0 'False
    Width          =5895

Object: PictureBox
Object Name: ColorBar
    ClipControls   =0 'False           Height        =270
    Left           =135                Picture       =Visual Basic 256 Palette
    ScaleMode      =0 'User            Top           =90 'within ColorMask
    Visible        =0 'False

Object: HScrollBar
Object Name: ColorScroll
    Height         =240                LargeChange   =96
    Left           =6240               Max           =0
    Min            =-800               SmallChange   =32
    Top            =540 'within StatusArea  Visible  0 'False
    Width          =510
```

Object: CommandButton
Object Name: No_Fill

Caption	="X"	Height	=240
Left	=6180	Top	=180 ' within StatusArea
Visible	=0 ' False	Width	=585

Object: CommandButton
Object Name: TestButton

Caption	="test"	Height	=285
Left	=45	Top	=90 ' within StatusArea
Visible	=0 ' False	Width	=780

Object: Label
Object Name: XYLabel

Alignment	=2 ' Center	BackColor	=&H00FFFF00&
BorderStyle	=1 ' Fixed Single	Height	=315
Left	=900	Top	=60 ' within StatusArea
Visible	=0' False	Width	=5055

Object: PictureBox
Object Name: PictureMask

BackColor	=&H00008000&	BorderStyle	=0 ' None
ClipControls	=0 ' False	Height	=3975
Left	=780	ScaleMode	=3 ' Pixel
Top	=720	Visible	=0' False
Width	=4245		

Object: PictureClip
Object Name: PicClip1

Cols	=8	Location	="3840,1365,2475,240"
Picture	=ClipAPI. BMP	Rows	=3

Object: PictureBox
Object Name: ArtWork

AutoRedraw	=-1 ' True	BackColor	=&H00FFFFFF&
BorderStyle	=0 ' None	FontBold	=0 ' False
FontItalic	=0 ' False	FontName	="Arial"
FontSize	=12	FontStrikethru	=0 ' False
FontUnderline	=0 ' False	Height	=2145
Left	=180	ScaleMode	=Twips
Top	=120 ' within PictureMask	Visible	=0 ' False
Width	=3840		

Object: PictureBox
Object Name: ArtMask

AutoRedraw	=-1 ' True	BackColor	=&H00FFFFFF&
ClipControls	=0 ' False	DrawStyle	=6 ' Inside Solid
DrawWidth	=4	Height	=870
Left	=2220	ScaleMode	=Twips
Top	=180 ' within Artwork	Visible	=0 ' False
Width	=1425		

Object: PictureBox
Object Name: ScaleBox

BackColor	=&H00FFFFFF&	Height	=330
Left	=1395	ScaleHeight	=300
ScaleWidth	=300	TabIndex	=14
Top	=45	Visible	=0 ' False

```
    Width          =330
```

Object：PictureBox
Object Name：BezHandle 1

```
    BackColor      =&H00FF00FF&             BorderStyle    =0 'None
    ClipControls   =0 'False                DrawMode       =6 'Invert
    Height         =105                     Left           =45
    ScaleHeight    =105                     ScaleWidth     =105
    Top            =270 'within Artwork     Visible        =0 'False
    Width          =105
```

Object：PictureBox
Object Name：BezHandle 2

```
    BackColor      =&H00FF00FF&             BorderStyle    =0 'None
    ClipControls   =0 'False                DrawMode       =6 'Invert
    Height         =105                     Left           =45
    ScaleHeight    =105                     ScaleWidth     =105
    Top            =495 'within Artwork     Visible        =0 'False
    Width          =105
```

Object：PictureBox
Object Name：BezHandle 3

```
    BackColor      =&H00FF00FF&             BorderStyle    =0 'None
    ClipControls   =0 'False                DrawMode       =6 'Invert
    Height         =105                     Left           =45
    ScaleHeight    =105                     ScaleWidth     =105
    Top            =675 'within Artwork     Visible        =0 'False
    Width          =105
```

Object：PictureBox
Object Name：BezHandle 4

```
    BackColor      =&H00FF00FF&             BorderStyle    =0 'None
    ClipControls   =0 'False                DrawMode       =6 'Invert
    Height         =105                     Left           =45
    ScaleHeight    =105                     ScaleWidth     =105
    Top            =855 'within Artwork     Visible        =0 'False
    Width          =105
```

Object：Image
Object Name：Node 4 indexed controls

```
    BorderStyle    =1 'Fixed Single         Height         =90
    Index          =0 to 3                  Left           =840
    MousePointer   =2 'Cross                Top            =' within Artwork
    Visible        =0 'False                Width          =90
```

Object：Image
Object Name：handle 8 indexed controls

```
    Height         =120                     Index          =7
    Left           =' within Artwork        MousePointer   =2 'Cross
    Picture        =Handle.BMP              Top            =1170
    Visible        =0 'False                Width          =120
```

Object：Image
Object Name：Pnode

```
    BorderStyle    =1 'Fixed Single         Height         =510
    Index          =0                       Left           =' within Artwork
    MousePointer   =2 'Cross                Top            =45
    Visible        =0 'False                Width          =615
```

```
Object: Label
Object Name: Txt
      AutoSize        =-1 'True              BackColor      =&H00FFFFFF&
      BackStyle       =0 'Transparent        BorderStyle    =1 'Fixed Single
      Height          =480                   Left           =' within Artwork
      TabIndex        =5                     Top            =945
      Visible         =0 'False              Width          =15

Object: Shape
Object Name: Shape2
      BackColor       =&H00FF0000&           BackStyle      =1 'Opaque
      BorderStyle     =0 'Transparent        Height         =105
      Left            =45                    Top            =' within Artwork
      Visible         =0 'False              Width          =105

Object: Shape
Object Name: Shape1
      BackColor       =&H00FF0000&           BackStyle      =1 'Opaque
      BorderStyle     =0 'Transparent        Height         =105
      Left            =60                    Top            =' within Artwork
      Visible         =0 'False              Width          =105

Object: Menu
Object Name: Menu_File
      Caption         ="&File"

Object: Menu
Object Name: Menu_FileSelection
      Caption         ="&New"                Index          =0

Menu_FileSelection
      Caption         ="&Open..."            Index          =1

Menu_FileSelection
      Caption         ="&Save"               Index          =2

Menu_FileSelection
      Caption         ="Save &As..."         Index          =3
      Shortcut        ={F12}

Menu_FileSelection
      Caption         ="-"                   Index          =4

Menu_FileSelection
      Caption         ="&Print"              Index          =5

Menu_FileSelection
      Caption         ="-"                   Index          =6

Menu_FileSelection
      Caption         ="E&xit"               Index          =7

Object: Menu
Object Name:Menu_Arrange
      Caption         ="&Arrange"

Object: Menu
Object Name:Menu_ArrangeSelection
      Caption         ="To &Front"           Index          =0

Menu_ArrangeSelection
      Caption         ="To &Back"            Index          =1
```

```
Menu_ArrangeSelection
    Caption          ="Delete"              Index        =2

Object: Menu Menu_ArrangeSelection
    Caption          ="POPUP MENU"          Checked      =-1 'True
    Index            =3

Menu_ArrangeSelection
    Caption          ="ShowArtMask"         Index        =4

Object: Menu
Object Name:Menu_Lines
    Caption          ="&Lines"

Object: Menu
Object Name:Menu_SubLines
    Index            =0 to 7 blank captions

Object: Menu
Object Name:Menu_Paint
    Caption          ="&Paint"

Object: Menu
Object Name:Menu_SubPaint
    Index            =0 to 7 blank captions

Object: Menu
Object Name:Menu_Text
    Caption          ="&Text"

Object: Menu
Object Name:Menu_SubText
    Index            =0 to 20 blank captions
```

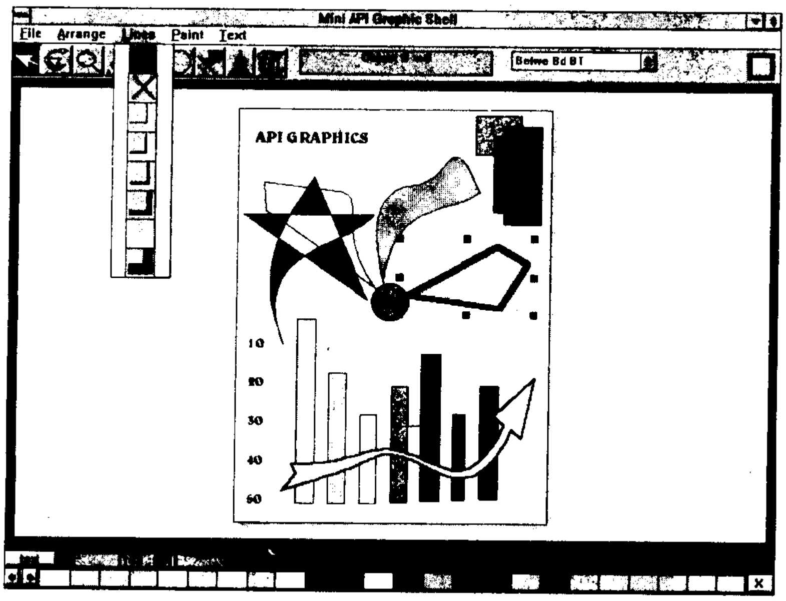

Mini API Graphic Shell
File
Arrange
Lines
Paint
Text
Belwe Bd BT
API GRAPHICS
10
20
30
40
60

图 15.2　ArtAPI 窗体的一幅屏幕图象

15.14 ArtAPI 窗体中新的声明

为了节省空间，ArtAPI 的声明部分只列出了全部变量和数组中的一部分。没有列出的 Dim 语句同第 2 章中 Artisan 程序的声明部分完全一样。清单 15.1 中只列出新的变量和数组。

清单 15.1 启动时新变量名的清单

```
Declarations
DefInt A－Z
Dim FlagPrinter As Integer
Dim FlagArtMask As Integer
Dim Foundcolor As Integer
Dim EraseOldImage As Integer
Dim UpdateMove As Integer
Dim MaskForeColor As Long
Const Limit = 100 'maximum number of objects you can draw
Dim B As Integer      ' Current Object (bounding box)
Dim Total As Integer      ' Total Number of Objects
Dim DrawOrder(Limit) As Integer
Dim Draw As Integer
Dim FillValue(Limit) As Integer 'No Fill
Dim LineValue(Limit) As Integer 'No Line
Dim OFFSETx, OFFSETy As Integer
Dim WIDE, HIGH As Integer 'Monitor Type
Dim PixelX, PixelY As Integer
Dim SSM As Single 'Side Scroll Move
Dim BSM As Single 'Bottom Scroll Move
Dim LastMove As Integer
Dim LastMenu As Integer
Dim FlagMove As Integer
''''Poly-shape variables''''
Dim NodeTotal As Integer 'Number of nodes per object
Dim FlagPoly(Limit) As Integer    'Polygon Redraw Flag
Dim FlagPolyLine(Limit) As Integer 'PolyLine Redraw Flag
Dim PolyPts As Integer    'API number of points
Dim Npt(0 To 300) As POINTAPI    'API address of each point
Dim FirstPolyPt(Limit) As Integer  'First Point of each object
Dim LastPolyPt(Limit) As Integer  'Last point of each object
Dim PolyShape(Limit) As Integer
Dim fp, LP, ZZ, Counter As Integer 'calculate above to pass to API
Dim Toggle As Integer
Dim PolyClose As Integer
Dim PolyCurveNode As Integer
Dim CountPolyPts As Integer
Dim WB, NB, SB, EB As Integer
Dim X1 As Long:Dim Y1 As Long:Dim X2 As Long:Dim Y2 As Long
Dim X3 As Long:Dim Y3 As Long:Dim X4 As Long:Dim Y4 As Long
Dim PtSize As Single
```

Dim FlagArtMask：用来在 ArtMask 控制对象中重绘 Bézier 曲线。（参见清单 15.22 中的 Poly_Curves 过程）。

Dim Foundcolor：决定 Find_Object 过程是否继续查找。

Dim EraseOldImage：当移动一多边形后清除旧的图象位置。

Dim MaskForeColor：规定了 ArtMask 重绘对象轮廓所用的颜色。

Dim DrawOrder(Limit)：以数组中记录的次序重绘所有对象。参见 Front_to_Back 过程。

Dim NodeTotal：统计每个加入多边形并显示的新节点。

Dim FlagPoly(Limit)：用于 UpdateObjects 过程来绘制一闭合多边形。

Dim FlagPolyLine(Limit)：用于 UpdateObjects 过程来绘制一开多边形。

Dim PolyPts：用于 API 函数 Polygon 和 Polyline 以指定要绘制的 X、Y 点个数。

Dim Npt(0 To 300)：用于 API 函数 Polygon 和 Polyline 以指定 X 和 Y 位置。如：

```
API=Polyline(Artwork.hDC;Npt(1),PolyPts)
```

折线绘制于 Artwork 图片框控制对象。X 和 Y 坐标存储于定义为 Npt 的 POINTAPI 型结构数组中，Npt(1)是第一个索引位置。PolyPts 表示要绘制的 X、Y 点数组。

Dim FirstPolyPt(Limit)：每个多边对象的第一个节点位置。用于定位其他节点。

Dim LastPolyPt(Limit)：每个多边对象的最后一个节点位置。用于定位其他节点。

Dim PolyShape(Limit)：查看当前对象是一多边形还是其他对象(如文本或矩形框)。

Dim fp,LP,ZZ,Counter：用以记录多边形如何构成的变量。

Dim Toggle：检查是否可以安全地画一折线。

Dim PolyClose：通知多边过程多边形已闭合并需要刷新。当最后一个节点和线段与第一个节点和线段相接时，即为多边形。三角形就是一个例子。

Dim PolyCurveNode：通知 Pnode 过程多边形中有相接的 Bézier 曲线。

Dim CountPolyPts：当移动一多边节点时，作为 Poly_Curves 过程(清单 15.22)清除头一次图象的标志。

Dim X1,Y1,X2,Y2：主要用于保持一条直线或两条折线的始末点坐标。

Dim X3,Y3,X4,Y4：当移动两条折线段的中间节点时所需的附加 **X** 和 **Y** 坐标。

Dim PtSize As Single：在当前字体点阵大小改变时保存字体尺寸。在 TxtSize_MouseUp 事件期间 PtSize 的值一直保存着。RedrawText 过程将 PtSize 的值置入 TextSize 数组，该数组在屏幕重绘和打印时给每个字串赋以当前字体点阵大小。

15.15 将折线转换为曲线

在大多数 ArtAPI 程序的事件和过程中，仅需对 Artisan 程序(从第 2 章到第 5 章)中的代码作轻微改动即可。很明显的变动是新的变量标志 PolyShape()和 FlagPolyLine。既然你已经把绘制连接和曲线的功能以多边形的形式集成起来，就应有一标志来通知程序把多边形当作特殊对象。对其他类型的对象，使用诸如 FlagBox，FlagCircle 及 FlagText 等变量标志。

FlagPolyLine 是被赋给多边形头一条线段的主要变量。三角形就是一个例子。三角形由三条独立的折线段组成，每条折线段都被赋以一个叫做 PolyShape()的变量名标记。

为了在将折线转换为曲线时赋以适当的值，就必须要有 PolyShape()标志。本程序也使

用 PolyShape()标志来在多边形周围放置图柄或将多边形对象移到新的屏幕位置。

一个多边形总是从一条线开始(FlagLine)，当加入一条线段后形成折线(FlagPolyLine)。如果多边形已闭合，则删去 FlagPolyLine 标志，然后在对象的属性表中加入 FlagPoly 标志。

清单 15.2 中的 Artwork_DblClick 事件判断是否双击了直线、Bézier 曲线或折线。头一个 Select Case True 语句块使转换按钮有效以备选择。当选择一按钮后，对象被转换为直线或曲线。下一个 Select Case True 语句块给对象赋以新的标志名。然后通过置 0 来删去原标志。

清单 15.2 将曲线转换为直线，或相反

```
Sub Artwork_DblClick ()
    If StatusLabel.Caption = "No Object Found" Then Exit Sub
    If StatusLabel.Caption = "No Color Found" Then Exit Sub
If B = FlagBox(B) Or FlagCircle(B) Or FlagText(B) Or 0 Then
    Message $ = "Convert to curves is set for Poly_shapes and Lines only."
    Message $ = Message $ + Chr(13) + "Skewing and rotation not available"
    MsgBox Message $
    Exit Sub
End If
Select Case True
    Case FlagLine(B) = B
        NodeEdit! To_Line.Enabled = False
    Case FlagBézier(B) = B
        NodeEdit! To_Curve.Enabled = False
    Case PolyShape(B) = FlagBezier(B)
        NodeEdit! To_Curve.Enabled = False
    Case Else
        NodeEdit! To_Line.Enabled = False
End Select
    If CurrentTool = 1 Then NodeEdit.Show Modal Else Exit Sub
    StatusLabel.Caption = "Object B = " + Format $ (B)
Select Case True
    Case Convert_Curve
        FlagLine(B) = 0: FlagBézier(B) = B
        BezErase = True: BezHandles
        BezHandle1.Visible = True: BezHandle2.Visible = True
        Convert_Curve = False: NodeEdit! To_Line.Enabled = True
    Case Convert_Line
        W(B) = p1X(B): N(B) = p1Y(B): E(B) = p2X(B): S(B) = p2Y(B)
        p1X(B) = 0: p1Y(B) = 0: p2X(B) = 0: p2Y(B) = 0
        If PolyShape(B) = B Then FlagLine(B) = 0 Else FlagLine(B) = B
        FlagBézier(B) = 0
        Nodes
        Convert_Line = False: NodeEdit! To_Curve.Enabled = True
        UpdateObjects
    End Select
End Sub
```

Select Case True：当选中一对象，则返回对象编号(B)。如果对象不是 Case 表达式中指

定的类型，其值为 False。

Case FlagLine，FlagBézier，PolyShape，Else：使能或禁止弹出式 NodeEdit 窗体中相应的'To Curve'或'To Line'按钮。

Select Case True：根据所选按钮，执行与其索引匹配的 Case 块。

Case Convert_Curve：当转换为曲线后，一条直线或折线需要赋以 FlagBézier 标志。

Case Convert_Line：将曲线转换为直线需要特别的处理，这取决于是单独一条 Bézier 曲线还是在一多边形对象中的 Bézier 曲线。如果是多边形中的曲线，它将保持 FlagPolyLine 状态，而不是改为新的 FlagLine 标志。旧曲线的所有 4 个控制点均清为 0。

图 15.3 显示了各种文本属性的效果。

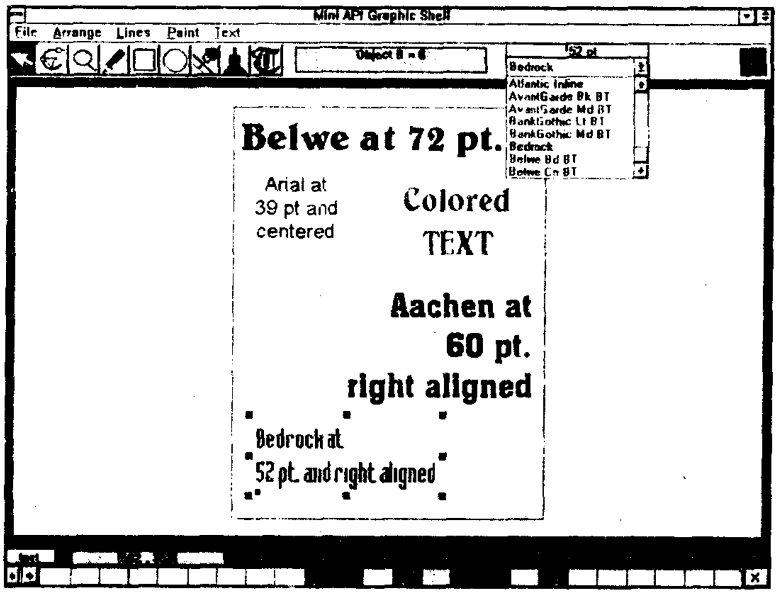

Mini API Graphic Shell
File
Arrange
Lines
Paint
Text
Object 8 = 6
52 pt
Bedrock
Atlantic Inline
AvantGarde Bk BT
AvantGarde Md BT
BankGothic Lt BT
BankGothic Md BT
Bedrock
Belwe Bd BT
Belwe Cn BT
Belwe at 72 pt.
Arial at
39 pt and
centered
Colored
TEXT
Aachen at
60 pt.
right aligned
Bedrock at
52 pt. and right aligned

图 15.3　不同文本属性的效果

15.16　Artwork 鼠标事件

清单 15.3 后面附有简要的代码说明，只解释了第 2 章 Artisan 的 MouseDown 事件以外新添的语句。

清单 15.3　在桌面上寻找一多边形

```
Sub Artwork_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
If PolyClose = True Then Pnode(0).Visible = False: PolyClose = False: Poly_Lines
If BezHandle3.Visible = False Then X1 = 0: Y1 = 0: X2 = X1: Y2 = Y1
''''''''

If Button = Left_Button Then
  ArtWork.DrawStyle = 0: ArtWork.DrawMode = 6
  N_H_Clear

Select Case CurrentTool
    Case PointerTool
        Find_Object X, Y
        BoundingBox_Down X, Y
    Case BézierTool
        If BezHandle1.Visible = True Then ShowDotted = False: BezUpdate
        If BezHandle3.Visible Then BezHandleMove Index
        Find_Object X, Y
        If B > 0 And PolyShape(B) = B Then Poly_Reload
        BoundingBox_Down X, Y
    Case LineTool To CircleTool
        BoundingBox_Down X, Y
    Case TextTool
        If txt.Caption > "" Then RedrawText' : Align = 0
        If Button = Left_Button Then
        txt.BorderStyle = 0
        Dim tm As TEXTMETRIC
        API = GetTextMetrics(ArtWork.hDC, tm)
        Leading = (tm.tmExternalLeading + tm.tmInternalLeading) * PixelY
        txt.Left = X: txt.Top = Y - (txt.Height - Leading)
    End If
End Select
End If
End Sub
```

If PolyClose=True Then:刚绘制了一个开的折线形并在桌面上击下鼠标。调用 Poly_Lines 过程隐藏节点并重绘新的开折线形。

Case BézierTool:当用户以鼠标接触多边形轮廓时,此新例程向多边形对象载入 Pnode 控制对象。

Find_Object X,Y:本过程(清单 15.11)判定鼠标是否接触多边形对象的轮廓。

If B>0 And PolyShape(B)=B Then:如果鼠标触及某多边形对象,B 将大于 0,同时多边形的对象编号将匹配 Find_Object 过程找到的对象。然后调用 Poly_Reload 过程在多边形周围载入节点。

15.17 在桌面上重绘多边形

清单 15.4 后面附有简要代码说明,它只解释了第 2 章 Artisan MouseUp 事件基础上新添的语句。

清单 15.4 使用 API 图形函数绘制对象

```
Sub Artwork_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
If CurrentTool <> 8 Then ArtWork.MousePointer = 0
PX = PixelX: PY = PixelY
If Button = Left_Button Then
Select Case CurrentTool
  Case PointerTool
   If UpdateMove = True Then
      MoveObject_Up Button, Shift, X, Y
      UpdateObjects
      UpdateMove = False: FlagMove = False: Exit Sub
   End If
```

```
        If MoveMouse() = True Then
            ArtWork.DrawStyle = Dot: ArtWork.ForeColor = Yellow
            API = Rectangle(ArtWork.hDC, X1 \ PX, Y1 \ PY, X2 \ PX, Y2 \ PY)
            FindAllObjects X1, Y1, X2, Y2, X, Y: Exit Sub
        Else
            If B > 0 Then Handles
        End If

      Case BézierTool
        If MoveMouse() = True Then
            ArtWork.DrawStyle = Dot: ArtWork.ForeColor = Yellow:
            API = Rectangle(ArtWork.hDC, X1 \ PX, Y1 \ PY, X2 \ PX, Y2 \ PY)
        End If
      Case LineTool To CircleTool
        If MoveMouse() = True Then
            BoundingBox_Up X1, Y1, X2, Y2
            If B = Total Then Draw = True: UpdateObjects
        End If

      Case TextTool
        txt.BorderStyle = 1: txt.Width = 15: txt.Visible = True
  End Select
        ArtWork.DrawWidth = 1: ArtWork.DrawStyle = Solid
        UpdateMove = False: FlagMove = False
  End If

  If Button = 2 And Menu_ArrangeSelection(3).Checked And CurrentTool = 0 Then
        Dim X_Y As POINTAPI
        GetCursorPos X_Y: TPM_CENTERALIGN = &H4: TPM_RIGHTBUTTON = &H2
        API = TrackPopupMenu(GetSubMenu(GetMenu(hWnd), LastMenu),
                TPM_CENTERALIGN, X_Y.X, X_Y.Y, 0, hWnd, 0)
  End If
  End Sub
```

PX＝PixelX:PY＝PixelY:在 PointerTool 和 BézierTool Case 表达式中的 API 图形函数要求以象素为坐标单位。在 Form_Load 事件期间 PixelX 和 PixelY 使用 Screen.TwipsPerPixel 方法得到正确的转换值。

If Button＝Left_Button Then:如果点下左鼠标按钮,执行与当前工具匹配的 Case 块。

Select Case CurrentTool:转向与当前工具匹配的 Case 块。

Case PointerTool:如果选择了指针工具(编号 1),执行后续语句块。

If UpdateMove＝True Then:只有在移动对象时 UpdateMove 变量才为真。在这种情况下,只需重绘移动了的对象。

MovePoly_Up Button,Shift,X,Y:如果将一多边形移动到桌面上新的位置,MoveObject_Up 事件刷新对象的所有 X,Y 坐标位置。

Case BézierTool,LineTool,etc. :在其他情况下,根据创建对象的工具绘制一个对象。本过程用 API 图形函数替换了 Visual Basic 的图形语句。在某些情况下,API 图形语句在 BoundingBox_Up 过程中执行。

图 15.4 到图 15.10 说明了创建一个多边形的步骤。

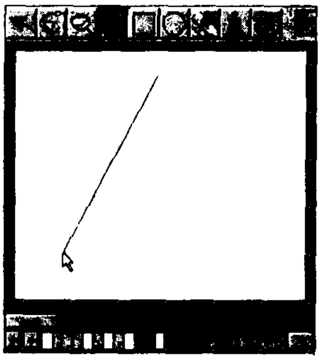


图 15.4　为了产生一多边形，选取画直线工具，绘制一条直线

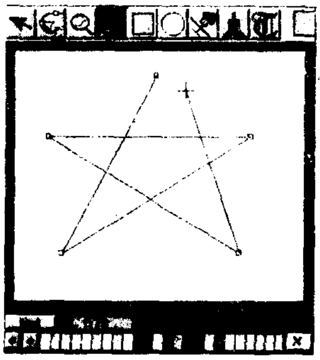


图 15.5　从每条线段的末结点连续画线形成一个多边形

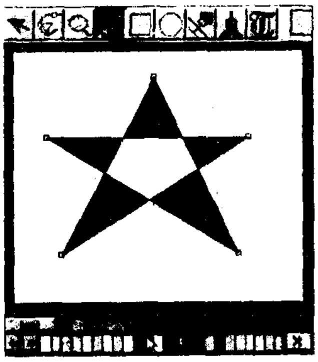


图 15.6　将最后一条线段连到第一个节点并选取一颜色填充多边形

15.18　Pnode 图象控制对象的 Tag 属性

几乎所有的 Visual Basic 控制对象都有一个 Tag 属性。Tag 是赋给窗体上的控制对象或窗体本身的一个字串表达式，可用来识别控制对象或将控制对象与窗体的信息传给其他过程、事件、控制对象或窗体。Visual Basic 语言参考手册中对此有详细介绍。

在 ArtAPI 程序中，有两种节点。第一种是一个图象控制对象，称为 Node，只用于直线、方框、圆及文本对象。第二种也是图象控制，叫做 Pnode(是 polynode，即多元节点的缩写)，

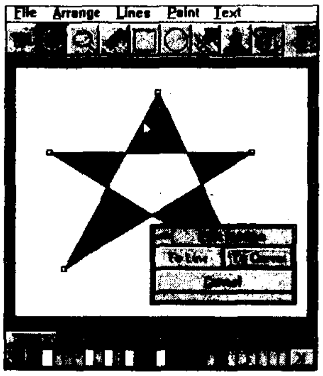

File
Arrange
Lines
Paint
Text

图 15.7　选择形状工具并双击一条直线使之转换为曲线

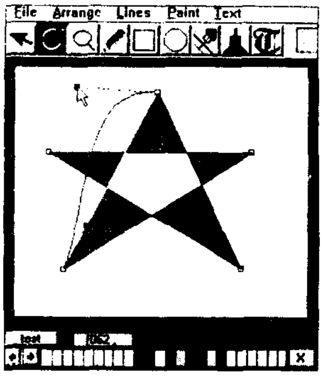

File
Arrange
Lines
Paint
Text

图 15.8　使用 Bézier 图柄编辑多边形

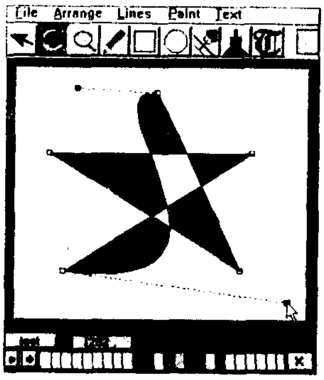

File
Arrange
Lines
Paint
Text

图 15.9　使用两个 Bézier 图柄来产生一个星形

只用于多边形的线段连接点。

你可以把 Node 的鼠标事件中的所有代码归入 Pnode 的鼠标事件。不过这就需要更长的 SelectCase 语句。

下面解释了在 ArtAPI 程序中可能赋给 Pnode 的三个标记名：

Pnode(Index).Tag：如果 Pnode 连接着一条 Bézier 曲线，那么将赋给 Pnode 图象控制对象一个 Tag。如果 Pnode 从左向右连接一条直线段和一条 Bézier 曲线，则标记此 Pnode

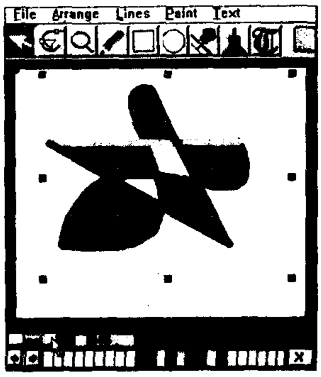


图 15.10　选择测试(Test)按钮来绘制多边形正确的路径轮廓

为"Rightcurve"，如果是从右向左连接的，则标记为"Leftcurve"。如果 Pnode 连接着两条 Bézier 曲线，则标记为"2curves"。

可以利用这个信息来重绘每个 Pnode 所连接的折线段(直线—曲线、曲线—直线或曲线—曲线)。

Pnode(Index).Tag="Leftcurve"：Pnode 左侧连有 Bézier 曲线。

Pnode(Index).Tag="Rightcurve"：Pnode 右侧连有 Bézier 曲线。

Pnode(Index).Tag="2curves"：Pnode 连接着两条 Bézier 曲线。

在 Artisan 程序(第 3 章的部分)中，使用了两个图片框控制对象 BezNode1 和 BezNode2 来移动 Bézier 曲线的端点。本程序删去了这些控制对象，但把它们的事件代码置入前述的三个过程 BezNode_Down，BezNode_Move 及 BezNode_Up 中。

BezNode_Down 过程由 Pnode MouseDown 事件触发。为调用 BezNode_Down 过程，Pnode 图象控制对象必须连接着多边形中的一条曲线段。

清单 15.5　Pnode MouseDown 事件的一个子过程

```
Sub BezNode_Down (Index%, Button%, Shift%, Lever%,
                                  r%, rr%, rrr%, FCP%, LCP%, X!, Y!)
    ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
    If BezHandle1.Visible = True Then
        ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), Yellow
        ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), Yellow
    End If
If BezHandle3.Visible = True Then
    X2 = BezHandle3.Left + 45: Y2 = BezHandle3.Top + 45
    X4 = BezHandle4.Left + 45: Y4 = BezHandle4.Top + 45
    ArtWork.Line (X1, Y1)-(X2, Y2), Yellow
    ArtWork.Line (X3, Y3)-(X4, Y4), Yellow
End If

BezHandleMove Index 'remove Bézier Handle if visible
''''redraw the dotted handle lines for Xor movement''''
LastIndex = (LastPolyPt(B) - FirstPolyPt(B)) - 1
    If Index = 0 Then ii = LastIndex Else ii = Index - 1
```

```
        If Index = LastIndex Then i = 0 Else i = Index + 1
        If FlagPolyLine(FCP) = FirstPolyPt(B) Then i = Index + 1
    Select Case Pnode(Index).Tag
        Case "Leftcurve"
    X1 = Pnode(Index).Left + 45: Y1 = Pnode(Index).Top + 45
    X2 = BezHandle1.Left + 45: Y2 = BezHandle1.Top + 45
    X3 = Pnode(i).Left + 45: Y3 = Pnode(i).Top + 45
    X4 = BezHandle2.Left + 45: Y4 = BezHandle2.Top + 45
        Case "Rightcurve"
    X1 = Pnode(Index).Left + 45: Y1 = Pnode(Index).Top + 45
    X2 = BezHandle2.Left + 45: Y2 = BezHandle2.Top + 45
    X3 = Pnode(ii).Left + 45: Y3 = Pnode(ii).Top + 45
    X4 = BezHandle1.Left + 45: Y4 = BezHandle1.Top + 45
        Case "2curves"
    X1 = Pnode(Index).Left + 45: Y1 = Pnode(Index).Top + 45
    X2 = BezHandle3.Left + 45: Y2 = BezHandle3.Top + 45
    X3 = Pnode(Index).Left + 45: Y3 = Pnode(Index).Top + 45
    X4 = BezHandle4.Left + 45: Y4 = BezHandle4.Top + 45
    End Select
    ArtWork.DrawMode = 6: ArtWork.DrawStyle = 0
    If FlagPolyLine(FCP) = FirstPolyPt(B) Then Poly_CurveMove2
                      Index, Button, Shift, Lever, X, Y
    If FlagPoly(FCP) = FirstPolyPt(B) Then Poly_CurveMove1
                      Index, Button, Shift, Lever, X, Y
        ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
        ArtWork.Line (X1, Y1)-(X2, Y2), Yellow
        ArtWork.Line (X3, Y3)-(X4, Y4), Yellow
    End Sub
```

If Beznode1. Visible:If BezHandle3. Visible:连接着 Bézier 曲线控制点及图柄的虚线可见，故消除掉。

BezHandleMove Index:调用此过程(见 ARTAPI1.FRM 的源码)可移动、重新调节 Bézier 曲线的两个图柄或使之可视。Pnode 图象控制对象的 Tag 名称决定了使用哪个 BezHandle 控制对象。

现在重绘图柄虚线，即可用异或(Xor)方法显示出目标的移动。

LastIndex,FlagPolyLine(FCP),etc.:当每条折线第一次绘制时，本程序为多边形的第一点及最后一点指定一对象编号。此编号存放于数组 FirstPolyPt 和 LastPolyPt 中。利用这些信息，就可通知计算机现在移动的是哪个 Pnode，或者在现在情况下，向此节点画一条虚线。载入的每个 Pnode 都有一索引值。索引 0 对应着第一个 Pnode，然后向上计数，直到向多边形中放入最后一个 Pnode。通过加减某对象编号所对应的 FirstPolyPt 或 LastPolyPt 中的索引值，即可找到多边形中所有 Pnode 图象控制对象。

FlagPolyLine(FCP)告知计算机当前对象是否是开或闭多边形以便相应调整索引变量。不然的话，一个开多边形就会错误地在始末点具有虚线的图柄。

FCP 表示第一个控制点(First Control Point)，等于 FirstPolyPt(object #)。object # 为对象编号。

If FlagPolyLine (FCP)=FirstPolyPt(B) Then Poly_CurveMove2:调用此过程以计算

多边形端点数目并传给 API 函数 Polyline，此函数将清除旧的图象。仅当对象是开多边形时才可调用 Poly_CurveMove2 过程。（例如：字母 W 就是由 4 条线段组成的开多边形）。

If FlagPoly(FCP)=FirstPolyPt(B) Then Poly_CurveMove1：本过程与 PolyLineMove2 过程几乎完全相同，不同之处是新添了处理闭合多边形的语句（例如：三角形就是由 3 条线段组成的闭合多边形）。因为 Poly_CurveMove2 和 Poly_CurveMove1 过程使用了许多相同的代码例程，所以可将二者合并到一个过程中。

当移动 Pnode 时，使用清单 15.6 中的 BezNode_Move 过程绘制 Bézier 曲线的图柄控制线。BezNode_Move 过程可分支到另外的过程以绘制正在编辑或移动的多边形。

清单 15.6 Pnode MouseMove 事件的一个子程序

```
Sub BezNode_Move (Index%, Button%, Shift%, Lever%,
                                    r%, rr%, rrr%, FCP%, LCP%, X!, Y!)
If Button Then
    ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
Select Case Pnode(Index).Tag
    Case "Leftcurve", "Rightcurve"
        ArtWork.Line (X1, Y1)-(X2, Y2), Yellow
        X1 = X + Pnode(Index).Left
        Y1 = Y + Pnode(Index).Top
    Case "2curves"
        ArtWork.Line (X1, Y1)-(X2, Y2), Yellow
        ArtWork.Line (X3, Y3)-(X4, Y4), Yellow
        X1 = X + Pnode(Index).Left
        Y1 = Y + Pnode(Index).Top
        X3 = X + Pnode(Index).Left
        Y3 = Y + Pnode(Index).Top
End Select
''''''''''''''''''''''''''''''''''''''''''
If FlagPolyLine(FCP) = FirstPolyPt(B) Then Poly_CurveMove2
              Index, Button, Shift, Lever, X, Y
If FlagPoly(FCP) = FirstPolyPt(B) Then Poly_CurveMove1
              Index, Button, Shift, Lever, X, Y
''''''''''''''''''''''''''''''''''''''''''
    ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
    ArtWork.Line (X1, Y1)-(X2, Y2), Yellow
    If Pnode(Index).Tag = "2curves" Then ArtWork.Line
              (X3, Y3)-(X4, Y4), Yellow
End If
End Sub
```

ArtWork.DrawMode:ArtWork.DrawStyle：将绘制模式改为 Xor（异或）方式，将绘图样式置为虚线，即可绘制并移动连接着 Bézier 图柄的控制线。

Select Case Pnode(Index)Tag：查看当前 Pnode 的标记字串，然后选择与此字串匹配的 Case 块。

Case "Leftcurve","Rightcurve"：将要移动的 Pnode 或者是先连着一条直线再连着一条曲线，或者是先连着曲线再连着直线。只需绘制一条连接着 BezHandle 图象控制对象的

可移动虚线(因为此时只移动一条 Bézier 曲线)。

ArtWork. Line(X1,Y1)－(X2,Y2),Yellow:以 Xor(异或)方式绘制一条黄色(Yellow)虚线。屏幕上虚线将呈现为亮兰(青)色。

X1＝X＋Pnode(Index). Left:Y1＝Y＋Pnode(Index). Top:移动虚线的 X1,Y1 坐标,使之看起来是沿着拖动鼠标的方向移动。Pnode 是图象控制对象,不具有 Scaleleft 或 Scaletop 属性,所以不能使用在 Artisan 程序中用于 BezNode1 和 BezNode2 事件的方法。每个 Pnode 图象控制对象的内部度量从控制对象左上角算起,左上角为(0,0)。每个 Pnode 图象控制对象都位于 Artwork 图片框中,所以其 Left 和 Top 属性采用桌面坐标。把 Pnode 的内部 X 坐标加上其 Left 属性值,把内部 Y 坐标加上其 Top 属性值即可得 Pnode 的精确绘制坐标值。

Case"2curves":将要移动的 Pnode 连接着两条曲线。因为要移动两条 Bézier 曲线,故绘制两条连接到两个 BezHandle 图象控制对象的移动控制线。

ArtWork. Line(X1,Y1)－(X2,Y2),Yellow:绘第一条控制线。

ArtWork. Line(X3,Y3)－(X4,Y4),Yellow:绘第二条控制线。

X1,Y1,X3,Y3:移动两条控制线。

FCP 变量表示第一个控制点(First ControlPoint),即(FirstPolyPt(object＃))。object＃为对象编号。本标志告知程序对象是什么形状,即一个开或闭多边形。

If FlagPolyLine(FCP)... Then Poly_CurveMove2:调用此过程以绘制连接到被移动 Pnode 的两条折线。只有对象为开多边形时才调用 Poly_CurveMove2 过程(例如,字母 W 即为一个由 4 条线段组成的开多边形)。

If FlagPoly(FCP)... Then Poly_CurveMove1:本过程与 PolyCurveMove2 过程基本相同。不同点是代码只用于闭合多边形(例如,三角形即为一个由三条线段组成的闭多边形)。

ArtWork. DrawMode:ArtWork. DrawStyle:Poly_CurveMove2 与 Poly_CurveMove1 过程以 Solid(实心)及 Inverse(反转)模式绘制两条折线段。因此,应改回 Xor(异或)模式及虚线模式以重绘控制线。

ArtWork. Line(X1,Y1)－(X2,Y2),Yellow:重绘控制线。

If Pnode(Index). Tag＝"2curves" Then:重绘第二条控制线。

15.19　更新曲线的坐标

清单 15.7 中的 BezNode_Up 过程分为两个部分。第一部分用来更新连接着一条或两条曲线并被移到新位置的 Pnode 的坐标。第二部分处理连接着开或闭多边形中的折线的 Pnode。

移动 Pnode 后,将产生很长的坐标及变量标志清单,并且在 Beznode_Up 过程将执行彻底的清除操作。你必须执行五个操作:

1. 在合适的数组中保存对象的新位置。例如:如果你移动了一条 Bézier 曲线,则应更新此曲线的始末点(P1X,P1Y 和 P2X,P2Y)。如果某一条折线段是直线,还应更新每条线段的 W,N 和 S,E 始末点。
2. 使用 BézierStore 过程计算多边形的新坐标,包括 BezHandles 的位置。这样即可以绕

着多边形的边界框放置图柄。

3. 调用 Poly_Handles 过程来设置边界框尺寸。Poly_Handles 过程在重绘图形前先清除下面的图象区域。
4. 使用 UpdateObjects 过程重绘对象。
5. 重绘连接到所有 Bézier 图柄的虚线。

图 15.11 显示了一个多边形的节点和图柄的名称及放置次序。最上方的例图显示了一个由 4 条折线(1 到 4)及连接着每条线段的 Pnode 组成的多边形。中间的例图显示了 Bézier 曲线的控制点(P1,P2)的位置及 Bézier 曲线的图柄(H1,H2)的位置,此控制点及图柄在折线 2 和 4 转换为曲线后将连接至控制线。最下方的例图显示了当所有折线转换为曲线后曲线控制对象、图柄点控制线的位置。

清单 15.7　Pnode MouseUp 事件的一个子过程

```
Sub BezNode_Up (Index%, Button%, Shift%, Lever%, X!, Y!)
    CountPolyPts = 0: PolyPts = 0
    Pnode(Index).Left = X + Pnode(Index).Left - 45
    Pnode(Index).Top = Y + Pnode(Index).Top - 45
r = FirstPolyPt(B) + Index     'Index +1
rr = FirstPolyPt(B) + Index - 1     'Index
rrr = FirstPolyPt(B) + Index - 2 'Index -1
FCP = FirstPolyPt(B)     'First object
LCP = LastPolyPt(B) - 1     'last object
LastIndex = (LastPolyPt(B) - FirstPolyPt(B)) - 1
''''''''''''''''''''''''''''''''''''''''
If Pnode(Index).Tag = "2curves" Then
    Select Case Index
        Case 0
            p1X(FCP) = Pnode(Index).Left + 45
            p1Y(FCP) = Pnode(Index).Top + 45
            p2X(LCP) = Pnode(Index).Left + 45
            p2Y(LCP) = Pnode(Index).Top + 45
                B = FCP
                BézierStore
            B = LCP
p1.X    p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
            BézierStore
        Case Else
            p1X(r) = Pnode(Index).Left + 45
            p1Y(r) = Pnode(Index).Top + 45
            p2X(rr) = Pnode(Index).Left + 45
            p2Y(rr) = Pnode(Index).Top + 45
            B = r
            BézierStore
            B = rr
p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
            BézierStore
End Select
```

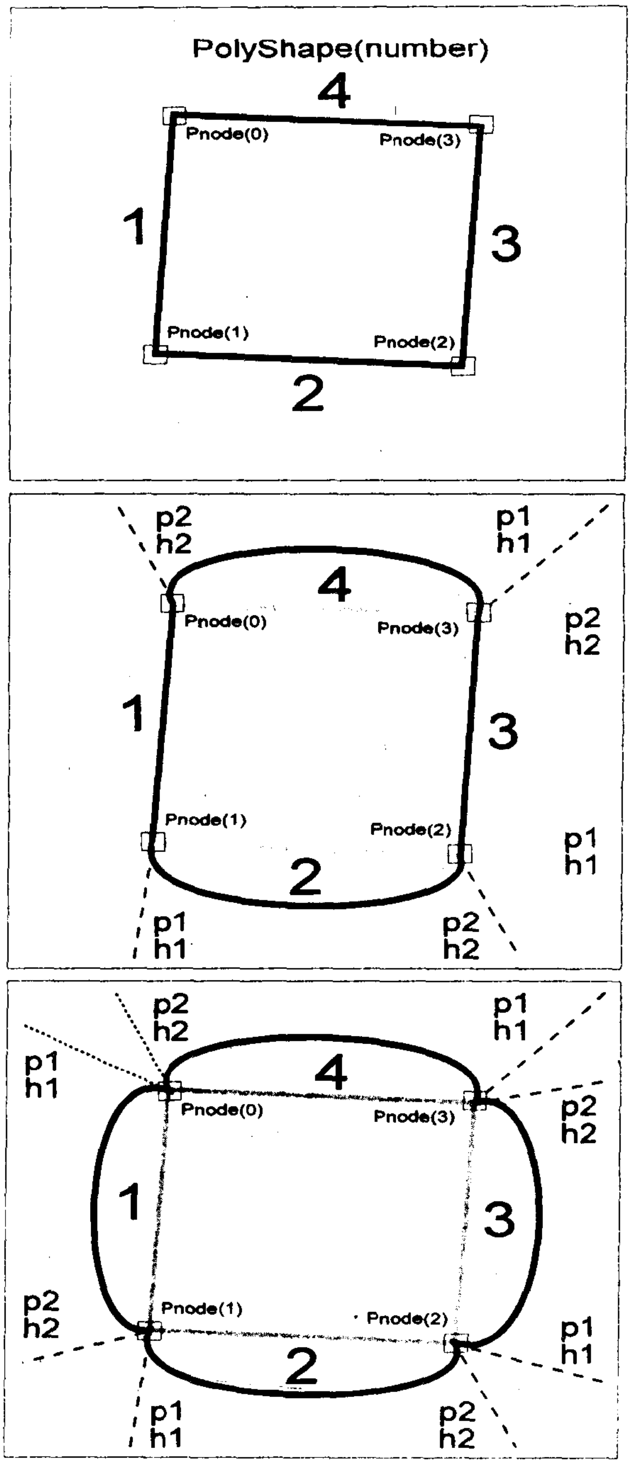

PolyShape(number)
1
2
3
4
Pnode(0)
Pnode(1)
Pnode(2)
Pnode(3)
p1
h1
p2
h2

图 15.11　多边形上直线和曲线的节点和图柄

下面是对 BezNode_Up 过程代码的逐行解释：

CountPolyPts=0：本变量为清除原折线段的标志。置为 0。

PolyPts=0：本变量记录着使用 API Polyline 语句时 X，Y 点的个数。置为 0。

Pnode(Index).Left，Pnode(Index).Top：将 Pnode 图象控制对象重新定位到新的坐标。

r，rr，rrr，FCP，LCP，LastIndex：这些变量用来找到刚移动的 Pnode 的索引号，同时考虑到了开和闭多边形的情况。

If Pnode(Index).Tag="2curves" Then：刚移动的 Pnode 连接着两条 Bézier 曲线，故必须执行以下语句：

Select Case Index：如果 Pnode 索引号为 0，则更新连接两条 Bézier 曲线的 P1（第一个控制点）索引点与 P2（第二个控制点）索引点。

B=FCP：要更新两条 Bézier 曲线，所以给 B 赋以第一个控制点（FCP），即 Pnode(0)左侧的 Bézier 曲线。

BézierStore：本过程寻找第一条 Bézier 曲线的边界框。这是因为 ArtMask 图片框要知道所有的多边形及边界框的尺寸大小，否则将不可能在 ArtAPI 程序中寻找或移动任何对象。

B=LCP：已找到第一条 Bézier 曲线的边界框，现在寻找第二条曲线的边界框。第二条曲线位于 Pnode(0)右侧，亦即最后一个控制点（LCP）对象。

p1. X= p1X(B), etc.：将这些变量改变为反映第二条曲线的控制点的值，以便由 BézierStore 过程用来寻找该曲线的边界框。

BézierStore：保存第二条 **Bézier** 曲线的边界框大小。

Case Else：除了刚移动的 Pnode 索引号大于 0 以外，本语句块同 Case 0 语句块完全相同。r 和 rr 变量表示了刚移动的 Pnode。

你现在应做的只是更新图案、在多边形周围放置边界框并重绘控制线。

清单 15.8 列出了 BezNode_Up 过程的第二部分。

清单 15.8　BezNode_Up 过程（续）

```
    Pnode(Index).Tag = ""
    EraseOldImage = True
        Poly_Handles
        UpdateObjects
    ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
X1 = Pnode(Index).Left + 45: Y1 = Pnode(Index).Top + 45
X2 = BezHandle3.Left + 45: Y2 = BezHandle3.Top + 45
X3 = Pnode(Index).Left + 45: Y3 = Pnode(Index).Top + 45
X4 = BezHandle4.Left + 45: Y4 = BezHandle4.Top + 45
    ArtWork.Line (X1, Y1)-(X2, Y2), Yellow
    ArtWork.Line (X3, Y3)-(X4, Y4), Yellow
Exit Sub
End If
''''''''''''''''''''''''''''''

If Index = 0 Then
    If FlagPolyLine(B) = B Then
    W(FCP) = Pnode(Index).Left + 45:N(FCP) = Pnode(Index).Top + 45
Else
    W(FCP) = Pnode(Index).Left + 45:N(FCP) = Pnode(Index).Top + 45
    E(LCP) = Pnode(Index).Left + 45:S(LCP) = Pnode(Index).Top + 45
End If
Else
    If FlagPolyLine(FCP) = FCP And LastPolyPt(B) = B + 1 Then
        E(rr) = Pnode(Index).Left + 45
        S(rr) = Pnode(Index).Top + 45
        Else
            W(r) = Pnode(Index).Left + 45
            N(r) = Pnode(Index).Top + 45
            E(rr) = Pnode(Index).Left + 45
            S(rr) = Pnode(Index).Top + 45
        End If
    Select Case Lever
        Case 1 'Beznode1_Up
                BezHandle1.ScaleLeft = BezHandle1.Left
            BezHandle1.ScaleTop = BezHandle1.Top
        Case 2
                BezHandle2.ScaleLeft = BezHandle2.Left
            BezHandle2.ScaleTop = BezHandle2.Top
    End Select
    p1X(B) = p1.X: p1Y(B) = p1.Y: p2X(B) = p2.X: p2Y(B) = p2.Y
    h1X(B) = h1.X: h1Y(B) = h1.Y: h2X(B) = h2.X: h2Y(B) = h2.Y
        EraseOldImage = True
            Poly_Handles
            UpdateObjects
    p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
    h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
            ArtWork.DrawMode = 7: ArtWork.DrawStyle = 2
    ArtWork.Line (p1.X, p1.Y)-(h1.X, h1.Y), Yellow
        ArtWork.Line (p2.X, p2.Y)-(h2.X, h2.Y), Yellow
            BézierStore
    End Sub
```

Pnode(Index).Tag="":清除 Pnode 的 2curve 标记。因为为了节省空间,总是重新载入 Pnode,所以正确的标记名称在 Pnode_MouseDown 事件中被指定。

EraseOldImage=True:本变量告知 Poly_Handle 过程必须清除旧图象的边界框并将新的边界框大小定为 W(0),N(0),E(0)和 S(0)。

Poly_Handles:为多边形的新边界框赋以坐标。

UpdateObjects:重绘桌面上所有对象。

ArtWork. DrawMode;**ArtWork. DrawStyle**:修改为 Xor(异或)方式和点线模式在新位置绘制虚线。

X1,X2,X3,X4:设置虚线的始末点。

ArtWork. Line:绘制这两条虚线。

Exit Sub:已更新了连接着两条曲线的一个 Pnode,退出此过程。

代码段的最后部分处理只连着一条曲线的 Pnode。下述代码同时也考虑了是否该曲线属于开或闭多边形。另外,如果曲线位于第一个或最后一个 Pnode 上,还要执行附加的例程。

If Index=0 Then:多边形中的第一个 Pnode 是特殊情况。If/Then 代码块必须检查第一个 Pnode 是否位于一个开或闭多边形中。

If FlagPolyLine(B)=B Then:开多边形的第一个 Pnode 没有要更新的终止点(E,S)(因为在拖动 Pnode(0)时 E 和 S 坐标保持不变)。

Else:第一个 Pnode 连接着闭多边形的始末线段。更新所有可能的线段的始末点(W,N, E 和 S)。

Else:更新索引号大于 0 的 Pnode。

If FlagPolyLine(FCP)=FCP And LastPolyPt(B)=B+1:开多边形的最后一个 Pnode 没有要更新的起始点(W,N)(因为在拖动最后一个 Pnode 时 W 和 N 坐标保持不变)。

Else:更新一个索引号大于 0 且位于闭多边形上的 Pnode。

W(r),N(r),E(rr),S(rr):更新闭多边形中连接着索引号大于 0 的 Pnode 的线段。

Select Case Lever:控制线由连接到 Bézier 曲线末端的虚线图柄组成。控制线 1 通常指定给 Pnode 图象控制对象的左侧部分或 Bézier 曲线的第一个控制点。控制线 2 通常指定给 Pnode 图象控制对象的左侧部分或 Bézier 曲线的末端控制点。控制线类似于第 3 章中 Artisan 程序的 BezNode1 和 BezNode2 控制对象。控制线可区别 p1X,p1Y 和 p2X,p2Y 位于曲线的哪一侧。

Case 1:移动了一个只连着一条 Bézier 曲线的 Pnode。此节点通常连接着曲线的起始点。在 Pnode_MouseMove 事件中 Lever 编号(1)被指定。

BezHandle1.ScaleLeft,ScaleTop,Visible:重置同 Lever1 相关的 BezHandle。

Case 2:移动了一个只连接着一条 Bézier 曲线的 Pnode,此节点通常连接着曲线的终止点。在 Pnode_MouseMove 事件中 Lever2 被指定。

BezHandle2. ScaleLeft,ScaleTop,Visible:重置同 Lever2 相关的 BezHandle。

p1X(B)=p1. X,etc. :此 Pnode 只连着一条曲线,因此可以很简单地更新此曲线的坐标点。p1. X,p1. Y,p2. X 和 p2. Y 保存着曲线的始末点坐标,故将这些值保存到合适的数组中。

EraseOldImage:Poly_Handles:UpdateObjects:清除旧的图象区,计算边界框坐标,然后重绘多边形。

p1. X=p1X(B),etc. :当使用 Updateobjects 重绘桌面上所有对象后,可将 p1X,p1Y,p2X 和 p2Y 还原为初始值。

BézierStore:保存多边形的新边界框位置。

15.20 移动对象

清单 15.9 中的 BoundingBox_Down 过程类似于第 2 章 Artisan 程序中的同名过程。主要的不同点是在 ArtAPI 程序的 BoundingBox_Down 过程中,要计算可移动的多边形边界框及可清除的边界框区域。

清单 15.9 中的 BoundingBox_Down 过程显示了移动对象的另一种方法。Movetest 程序(第 1 章)和 Artisan 程序(第 2 章)使用了一个公式,通常在移动过程中计算每一个点的位置来移动对象(在移动对象时修改变量)。本过程的结构和前述采用公式时相同,但是通过在 MouseUp 事件之后计算新位置可以减少代码量。

清单 15.9 是 Artwork_MouseDown 事件的子程序。

清单 15.9 Artwork_MouseDown 事件的一个子过程

```
Sub BoundingBox_Down (X As Single, Y As Single)
    X1 = X: Y1 = Y: X2 = X1: Y2 = Y1
    ArtWork.DrawMode = Inverse
If CurrentTool = 0 And ArtMask.Point(X - 15, Y - 15) = MaskForeColor Then
    If PolyShape(B) = B Then
        Poly_Handles 'Find Poly_Shape Bounding Box
        XX1 = W(0): YY1 = N(0): XX2 = E(0): YY2 = S(0)
        WB = W(0): NB = N(0): EB = E(0): SB = S(0)
    Else
        XX1 = W(B): YY1 = N(B): XX2 = E(B): YY2 = S(B)
        WB = W(B): NB = N(B): EB = E(B): SB = S(B)
    End If
    FlagMove = True: Foundcolor = False
End If
End Sub
```

当拖动对象（MouseMove）并释放（MouseUp）时，由 MoveObject（清单 15.16）和 MoveObject_Up(清单 15.17)来重新定位对象的所有新 X 和 Y 坐标点。

If CurrentTool=0 And ArtMask.Point(X-15,Y-15)：当前工具必须是指针工具，并且位于内存位图 ArtMask 中的象素必须与鼠标的当前 X 和 Y 位置相匹配。内存位图的 X、Y 点必须是 ArtMask.ForeColor，即为 0(黑色)。

If PolyShape(B)=B Then：对象是一个多边形。

Poly_Handles：计算一个多边形的边界框尺寸。图柄通常放置在边界框周围以便拉伸对象。

XX1=W(0):WB=W(0)：保存多边形的边界框位置。

Else XX1=W(B):WB=W(B)：保存一个矩形、直线、圆或文本的边界框位置。

FlagMove=True：通知程序已经准备好在用户拖动对象时移动此对象。

Foundcolor=False：Find_Object 过程已找到一对象。则重置此变量为 False。

15.21　API 绘图函数

BoundingBox_Move 过程(见清单 15.10)同 Artisan 程序中的过程不同，它使用 API 图形函数来绘制直线、方框和圆。在替换 Basic 图形语句时，必须遵守影响 API 函数的一些规则，主要的规则是 API 图形要求以象素单位给出坐标。另外，Visual Basic 使用其自定义的定标例程，当切换定标模式或设置内部定标尺寸时，会影响窗体或图片框中图形的放置，因此，本程序中删去了缩放和定标例程。在加入缩放特性之前最好先用第 14 章中的 Scaltest 程序试验一下。

清单 15.10　Artwork_MouseMove 事件的一个子过程

```
Sub BoundingBox_Move (Shift As Integer, X As Single, Y As Single)
If CurrentTool = 0 And FlagMove = True And MoveMouse() = True
                  Then MoveObject Button, Shift, X, Y: Exit Sub
    PX = PixelX: PY = PixelY
If MoveMouse() = True Then
  Select Case CurrentTool
    Case PointerTool To BezierTool
        ArtWork.DrawMode = Xor_Pen
```

```
            ArtWork.DrawStyle = Dot: ArtWork.ForeColor = Yellow
            r = Rectangle(ArtWork.hDC, X1 \ PixelX, Y1 \ PixelY,
                      X2 \ PixelX, Y2 \ PixelY)
        Case LineTool
            Npt(1).X = X1 \ PX: Npt(1).Y = Y1 \ PY
            Npt(2).X = X2 \ PX: Npt(2).Y = Y2 \ PY: PolyPts = 2
            r = Polyline(ArtWork.hDC, Npt(1), PolyPts)
        Case BoxTool
            r = Rectangle(ArtWork.hDC, X1 \ PixelX, Y1 \ PixelY,
                      X2 \ PixelX, Y2 \ PixelY)
        Case CircleTool
            r = Ellipse(ArtWork.hDC, X1 \ PixelX, Y1 \ PixelY, X2 \ PixelX,
                      Y2 \ PixelY)
        End Select
    End If
End Sub
```

If CurrentTool=0... Then MoveObject Button:在移动一个对象时,不需要绘制任何对象。

Select Case CurrentTool:转向适当的 API 图形函数以绘制与当前工具相联系的对象。

你也许希望在 LineTool Case 块中使用 API 的 LineTo 和 MoveTo 函数来画线。在第 8 章可找到大部分 Windows 3.1 API 绘图函数的示例。

本程序中的 API 图形调用使用 PixelX 和 PixelY 来把 twip 转换为象素。PixelX 和 PixelY 通过使用 TwipsperPixelX 和 TwipsperPixelY 屏幕数组来在 Form_Load 事件中设置(见 ARTAPI1.FRM 的源码)。在大多数 VGA 显示器上,垂直方向和水平方向每英寸各有 96 个象素。以每英寸 1440 twip 除以每英寸 96 象素,即得 TwipPerPixel 为 15。

此数值根据显示类型会有变化。可用 API 函数 GetDeviceCaps 来得到显示器的每英寸象素分辨率。

清单 15.11 中 BoundingBox_Up 过程记录着已绘制的对象数目,同时也为直线、方框及圆对象指定标志名。

清单 15.11　Artwork_MouseUp 事件的一个子过程

```
Sub BoundingBox_Up (X1&, Y1&, X2&, Y2&)
If MoveMouse() = True Then
    Total = Total + 1: B = Total
    DrawOrder(B) = Total
    StatusLabel.Caption = "Object B = " + Format$ (B)
    LineWidth(B) = ArtWork.DrawWidth
    W(B) = X1: E(B) = X2: N(B) = Y1: S(B) = Y2
Select Case CurrentTool
    Case LineTool  'Lineto
        FlagLine(B) = B:Nodes
    Case BoxTool   'Rectangle
        FlagBox(B) = B: FillValue(B) = 1:Nodes
    Case CircleTool    'Ellipse
        FlagCircle(B) = B:FillValue(B) = 1:Nodes
End Select
End If
End Sub
```

If MoveMouse()=True Then：如果鼠标移动了一段指定距离则继续执行。

Total=Total+1：B=Total：对象计数加 1。

DrawOrder(B)=Total：将赋给此对象的编号插入 DrawOrder 数组。To_Front_Back 过程（清单 15.31）可得到赋给 DrawOrder 数组的编号并打乱绘图次序，这样 UpdateObject 过程可将一个对象置于其余对象的前面或后面。

Select Case CurrentTool：给新对象赋以编号，然后在对象周围放置节点。

15.22　寻找对象的轮廓

在 Artisan 程序中，无法选中正好位于另一个对象下面的对象。鼠标只选择边界框而非对象的轮廓，这样就很难高亮显示当前对象。清单 15.12 中的 Find_Object 过程显示了通过鼠标接触来选择对象的一种方法。

FindObject 过程使用了第 2 章的 Artisan 程序中没有出现过的两个查找特性。第一个特性是 API 函数 PtInRect，以此来替代人工搜索对象边界框的工作；第二个查找特性为使用了一个叫做 ArtMask 的隐藏图片框。当在某对象的边界框中击下鼠标时，在 ArtMask 中会创建该对象的被屏蔽的轮廓图象。如果鼠标的坐标与对象的被屏蔽轮廓的坐标相同，则选中此对象。

清单 15.12　寻找桌面上任何对象的辅助线或轮廓线

```
Sub Find_Object (X As Single, Y As Single)
For r = -90 To 90 Step 15: For rr = -90 To 90 Step 15
    If ArtWork.Point(X - r, Y - rr) = ArtWork.BackColor
              Then Z = Z + 1 Else Foundcolor = True
    If Z = 169 Then StatusLabel.Caption = "No Color Found": B = 0: Exit Sub
Next rr: Next r
''''''''''''Find Bounding Box
I = Total
XY& = X + CLng(Y) * &H10000
Do While I > 0
Dim BB1 As Rect: Dim BB2 As Rect: Dim BB3    As Rect: Dim BB4 As Rect
BB1.Left = W(I) - 30: BB1.Top = N(I) - 30: BB1.Right = E(I) + 30
BB1.Bottom = S(I) + 30
BB2.Left = W(I) - 30: BB2.Top = S(I) - 30: BB2.Right = E(I) + 30
BB2.Bottom = N(I) + 30
BB3.Left = E(I) - 30: BB3.Top = S(I) - 30: BB3.Right = W(I) + 30
BB3.Bottom = N(I) + 30
BB4.Left = E(I) - 30: BB4.Top = N(I) - 30: BB4.Right = W(I) + 30
BB4.Bottom = S(I) + 30
If PtInRectBynum%(BB1, XY&) Or PtInRectBynum%(BB2, XY&)
    Or PtInRectBynum%(BB3, XY&) Or PtInRectBynum%(BB4, XY&) Then
B = i: ArtMask.Cls
If N(I) <= S(I) Then AT = N(I): AH = S(I) Else AT = S(I): AH = N(I)
If W(I) <= E(I) Then AL = W(I): AW = E(I) Else AL = E(I): AW = W(I)
ArtMask.Top = AT - 150
ArtMask.Left = AL - 150
ArtMask.Width = (AW - AL) + 300
ArtMask.Height = (AH - AT) + 300
ArtMask.ScaleLeft = ArtMask.Left
ArtMask.ScaleTop = ArtMask.Top
UpdateArtMask
    If ArtMask.Point(X - 15, Y - 15) = MaskForeColor Then Exit Do
End If
    If I = 0 Then Exit Do Else I = I - 1
Loop
        B = I: StatusLabel.Caption = "Object B = " + Format $ (B)
        ArtApi.StatusColor.BackColor = OutLineColor(B)
        ArtApi.StatusPaint.BackColor = PaintColor(B)
    Nodes
End Sub
```

For r=－90 To 90 Step 15：因为 Find_Object 过程是被 MouseDown 事件调用的，所以需要判断鼠标是否位于某对象之上，若不是，则退出此过程。

ArtWork.Point(X－r,Y－rr)：扫描鼠标周围的 169 个象素，寻找与桌面的白底色不同的颜色，如果找不到某对象颜色，则退出此过程。

本过程的第二部分对寻找桌面上任何可能的垂直或水平辅助线。如果找到一条辅助线，则将之擦除以便移动到新的位置。

下述代码用来寻找一个边界框。

XY&=X+CLng(Y)*&H1000：把 X 和 Y 拼接为 32 位值。API 函数 PtInRect 要求 Long 型的 X 和 Y 数值。

Dim BB1,BB2,BB3,BB4 As Rect：API 函数 PtInRect 寻找定义为 Rect Type 结构(分量为左、上、右和下)的边界框。如果使用图柄翻转了某对象，则边界框坐标也跟着翻转。因此需要 4 个 Rect 边界框结构来检查每种可能的边界框位置。

If PointInRectBynum%：如果鼠标位于某边界框之上，则必须找出触及了哪个对象。注意为了传递 32 位 Long 型参数，使用了 API 函数 PtInRect 的 Alias 别名。

If (N(I)<=S(I) Then AT=N(I)：在隐藏的 ArtMask 图片框中重绘可能位于此区域的所有对象。如果 Artwork 的当前 X 和 Y 坐标与 ArtMask 中的象素颜色确为黑色，即找到了对象。

ArtMask.ScaleLeft,Top,Width,Height：为了在使用持久位图时节省内存空间，把每个图片框的大小调整到它所寻找的每个对象的实际大小。UpdateArtMask 过程向隐藏的 ArtMask 图片框绘制对象。为观察实际效果，可选择主菜单栏中的"Show ArtMask"子菜单项。

15.23 为多边形计算边界框

ArtAPI 程序中 Handle 事件的唯一改动就是在对象成为多边形时调用 Poly_Handles。如果对象确实是一个多边形，那么清单 15.13 中的 Poly_Handles 过程将扫描所有折线段和曲线段的尺寸并形成多边形的边界框。一旦创建了多边形的边界框，用来调整多边形大小的八个图柄就放置在边界框的周围。

清单 15.13 合并整个多边形

```
Sub Poly_Handles ()
Dim i As Integer
    StoreB = B: ii = FirstPolyPt(B)
WB = W(ii): NB = N(ii): EB = E(ii): SB = S(ii)
    For i = FirstPolyPt(ii) To LastPolyPt(ii) - 1
        If WB > W(i) Then WB = W(i)
        If NB > N(i) Then NB = N(i)
        If EB < E(i) Then EB = E(i)
        If SB < S(i) Then SB = S(i)
    Next i
     For i = FirstPolyPt(B) To LastPolyPt(B) - 1
        If WB > E(i) Then WB = E(i)
        If NB > S(i) Then NB = S(i)
        If EB < W(i) Then EB = W(i)
        If SB < N(i) Then SB = N(i)
    Next i
If ArtWork.DrawStyle = Dot Then W(0) = WB: N(0) = NB:
                 E(0) = EB: S(0) = SB:Exit Sub
If EraseOldImage = True Then
    EraseOldImage = False
    XX1 = WB: YY1 = NB: XX2 = EB: YY2 = SB: Exit Sub
Else
    W(0) = WB: N(0) = NB: E(0) = EB: S(0) = SB
    B = 0
    If CurrentTool = 0 Then Handles
    B = StoreB
End If
End Sub
```

WB=W(ii).etc.：在扫描前保存初始对象数。

For i=FirstPolyPt(ii) To LastPolyPt(ii)-1：所绘制的每个多边形的第一个对象记录着第一个和最后一个 Pnode 的位置。因此，可以找到从第一个到最后一个之间的所有对象。

If WB>W(i) Then WB=W(i)：比较多边形中每个多边对象的西侧（左侧）位置。一旦此过程扫描了每个坐标，则最左侧的点等于 W(i)，最右侧的点等于 E(i)，最上边等于 N(i)，最下边等于 S(i)。

If WB>E(i) Then WB=E(i)：如果对象已翻转，则切换所有数值。

过程中其余代码用来保存旧的图象区域，以便在拉伸或移动后清除这些区域。

15.24 打开与保存多边形文件

清单 15.14 列出了 mnuFileOpen 过程的主要部分。文件打开和保存命令的完整清单可参见 ARTAPI1.FRM 源代码文件中的 mnuFileOpen 和 mnuFileSave 过程。在 ArtAPI 程序中，重绘任何多边形只需 5 个附加变量。如果合并一些多边形例程，那么这个数目还可以减少。第 6 个变量保存绘图次序，因为使用的是新的 To_Front_Back 过程。DrawOrder(B)用在 UpdateObject 过程中。

清单 15.14　用以打开文件的新的文件读出命令

```
Sub mnuFileOpen ()
Line Input #1, B1: FlagPoly(B) = B1
Line Input #1, B1: FirstPolyPt(B) = B1
Line Input #1, B1: LastPolyPt(B) = B1
Line Input #1, B1: PolyShape(B) = B1
Line Input #1, B1: FlagPolyLine(B) = B1
Line Input #1, B1: DrawOrder(B) = B1
'see ARTAPI.FRM file for addition values
End Sub
```

Line Input #1,BI:本程序使用 Visual Basic 的 Input#语句来读出对象属性和坐标。也可以用其他 Visual Basic 文件 I/O 方法。

清单 15.15 列出了 mnuFileSave 过程的主要部分。文件打开和保存命令的完整清单可参见第 4 章的 Artisan 程序。

清单 15.15　用以保存文件的新的文件写入命令

```
Sub mnuFileSave ()
31 : Print #1, FlagPoly(B)
32 : Print #1, FirstPolyPt(B)
33 : Print #1, LastPolyPt(B)
34 : Print #1, PolyShape(B)
35 : Print #1, FlagPolyLine(B)
36 : Print #1, DrawOrder(B)
'see ARTAPI.FRM file for addition values
End Sub
```

Print #1, FlagPoly(B):本程序使用 Visual Basic 的 Print#语句来保存对象属性和坐标。也可使用其他 Visual Basic 文件 I/O 方法。

15.25　移动多边形对象

通过附加的 Move_Object_Up 过程,可以大大缩短 BoundingBox_Down 和 MoveObject 的长度。尽管第 2 章中的 Artisan 程序使用了移动中坐标改动,也仍然需要额外的变量。可以通过移动后更新所有坐标来删去这些变量。

第 2 章中的 BoundingBox_Down 过程(清单 2.12)计算对象的中心后以鼠标的 X 和 Y 坐标移动对象。所用公式如下:

```
XCenter=X-((W(B)\2)+(E(B))\2)
YCenter=Y-((N(B)\2)+(S(B))\2)
OFFSETx=XCenter:OFFSETy=YCenter
```

在第 4 章的 MoveObject 过程中(清单 4.25),OffsetX 和 OffsetY 保持不变,其初值在 MouseDown 事件中设置。在 MouseMove 事件中,使用下述公式移动所有边界框,从中心偏移到鼠标的 CurrentX 和 CurrentY 位置。

```
W(B)=(W(B)-OFFSETx)+XConter:N(B)=(N(B)-OFFSETy)+YCenter
E(B)=(E(B)-OFFSETx)+XConter:S(B)=(S(B)-OFFSETy)+YCenter
```

上述过程中，不需要特意更新边界框位置，因为相关值在拖动对象时已作了修改。

在 ArtAPI 程序中，通过添加 MouseUp 过程，明显地减少了 MoveObject 和 MoveObject_Up 过程移动一个对象所需的代码量。在 BoundingBox_MouseDown 事件中通过从 X1 和 Y1 中减去 X 和 Y，可以得到鼠标移动的距离。然后从边界框的原始位置中减掉鼠标走过的距离即可移动对象。清单 15.16 中的 MoveObject 过程计算了使用鼠标移动对象所需的偏移坐标。

清单 15.16　把对象移动到屏幕上新的位置

```
Sub MoveObject (Button As Integer, Shift As Integer, X As Single, Y As Single)
If PolyShape(B) = B Then i = 0 Else i = B
    If ArtWork.MousePointer = 0 Then
        ArtWork.DrawMode = 7: ArtWork.DrawWidth = 1
        N_H_Clear
        ArtWork.Line (W(i), N(i))-(E(i), S(i)), Yellow, B
        UpdateMove = True: ArtWork.MousePointer = 5
    End If
XCenter = X1 - X
YCenter = Y1 - Y
ArtWork.Line (WB, NB)-(EB, SB), Yellow, B
    WB = W(i) - XCenter
    NB = N(i) - YCenter
    EB = E(i) - XCenter
    SB = S(i) - YCenter
ArtWork.Line (WB, NB)-(EB, SB), Yellow, B
End Sub
```

If PolyShape(B)=B Then i=0：判断要移动的对象是否是多边形，如果是多边形，则以 0 号对象临时保存其边界框坐标。

If ArtWork.MousePointer=0：必须初始化某些数值以指示正在移动对象。由于在鼠标移动时要经常调用此过程，所以使用图片框的鼠标图标属性作为标志来绕过第一次移动鼠标后的 If/Then 语句。

UpdateMove=True:ArtWork.MousePointer=5：UpdateMove 标志通知其他鼠标事件正在移动一个对象。鼠标指针即改为 4 个箭头的方向图标。

XCenter=X1-X:YCenter=Y1-Y：如果对象是个圆，那么将修改中心坐标以反映新坐标。

ArtWork.Line(WB,EB)-(EB,SB)：在移动鼠标时绘制对象的边界框轮廓。

在移动一对象并松开鼠标按钮后，清单 15.17 中的 MoveObject_Up 过程保存该对象的新坐标位置。

清单 15.17 在移动后修改对象的所有坐标

```
Sub MoveObject_Up (Button As Integer, Shift As Integer, X As Single, Y As Single)
    If PolyShape(B) = B Then
        First = FirstPolyPt(B): Last = LastPolyPt(B) - 1
    Else
        First = B: Last = B
    End If
For i = First To Last
    XCenter = X - X1
    YCenter = Y - Y1
    W(i) = W(i) + XCenter
    N(i) = N(i) + YCenter
    E(i) = E(i) + XCenter
    S(i) = S(i) + YCenter
If FlagBezier(i) = i Then
    p1X(i) = p1X(i) + XCenter
    p1Y(i) = p1Y(i) + YCenter
    p2X(i) = p2X(i) + XCenter
    p2Y(i) = p2Y(i) + YCenter
    h1X(i) = h1X(i) + XCenter
    h1Y(i) = h1Y(i) + YCenter
    h2X(i) = h2X(i) + XCenter
    h2Y(i) = h2Y(i) + YCenter
End If
Next i
    If FlagText(B) = B Then OFFSET(B) = (OFFSET(B) - OFFSETx) + XCenter
End Sub
```

If PolyShape(B)=B Then：所移动对象是多边形，因此必须修改多边形中每条折线和曲线的坐标。每个多边形对象的节点坐标始于 FirstPolyPt(poly-shape#)，结束于 LastPolyPt(poly-shape#)，Poly-shape# 为多边形编号。

First=B: Last=B：对象不是多边形，For/Next 语句只执行一次。

For i= First To Last：如果对象是多边形，则遍历并修改多边形的所有边界框坐标以及多边形中所有曲线的控制点。

15.26 多边形的节点(Pnode)

本程序中有两种节点。第一种是 Image(图象)控制对象，用来连接及移动单线对象。第二种节点叫做 Pnode，用来连接及移动折线或折曲线对象。

多边形总是始于两端带有节点控制对象的单线对象。在用户从单线对象的最后节点继续引出第二条线后，该形状变为折线形，原来的两个节点被隐藏并被替换为 Pnode。

在 Node 图象控制对象的事件中没有包含错误处理程序。在绘制完一个多边对象后从某节点引出一条直线可能会把两个甚至所有对象连成一体。这样可产生不同的移动及合并分组效果，但并不推荐这样做。

清单 15.18 中的 Pnode_MouseDown 过程判断 Pnode 连接着何种对象(如折线或 Bezier 曲线)。然后修改 Pnode 的标记名以指明 Pnode 的哪一侧连接着曲线(Leftcurve，Rightcurve)，或者是连着两条曲线。此信息将传给 BezNode_Down 过程以便在编辑或移动

时绘制多边形。如果 Pnode 没有连接曲线，此过程将绘制相连的折线。

清单 15.18　在移动前设置多边节点坐标

```
Sub Pnode_MouseDown (Index%， Button%， Shift%， X As Single， Y As Single)
If CurrentTool = 3 And Index < NodeTotal - 1 Then Beep： Exit Sub
If CurrentTool = 3 And Index < LastPolyPt(B) Then Beep： Exit Sub
StatusLabel.Caption = "Node " + Format $ (Index) + " ( " +
                      Format $ (Pnode(Index).Left \ PixelX) + " - " +
                         Format $ (Pnode(Index).Top \ PixelY) + " )"
ArtWork.DrawStyle = 0： ArtWork.DrawMode = Inverse
XX2 = E(0)： YY2 = S(0)： XX1 = W(0)： YY1 = N(0)
''''''''''''''''''''''''''''''''''''''''''''''''''''''
r = FirstPolyPt(B) + Index              'Index +1
rr = FirstPolyPt(B) + Index - 1     'Index
rrr = FirstPolyPt(B) + Index - 2 'Index -1
FCP = FirstPolyPt(B)      'First object
LCP = LastPolyPt(B) - 1      'last object
''''''''''''''''''''''''''''''''''''''''''''''''''''''
PIL = Pnode(Index).Left + 45
PIT = Pnode(Index).Top + 45
BezErase = False
Select Case CurrentTool 'tools is either Bézier or Line tool
   Case BezierTool
Select Case Index 'Find the object number first
   Case 0
If p1X(FCP) = PIL And p1Y(FCP) = PIT Then B = FCP： Pnode(Index).Tag = "Leftcurve"
If p2X(LCP) = PIL And p2Y(LCP) = PIT Then B = LCP：
   Pnode(Index).Tag = "Rightcurve"
If p1X(FCP) = PIL And p2X(LCP) = PIL Then B = LCP： Pnode(Index).Tag = "2curves"
   Case Else
If p1X(r) = PIL And p1Y(r) = PIT Then B = r： Pnode(Index).Tag = "Leftcurve"
If p2X(rr) = PIL And p2Y(rr) = PIT Then B = rr： Pnode(Index).Tag = "Rightcurve"
If p1X(r) = PIL And p2X(rr) = PIL Then Pnode(Index).Tag = "2curves"
End Select
Select Case Pnode(Index).Tag
   Case "Leftcurve"， "2curves"
Pnode(Index).Visible = False： PolyCurveNode = True
BezNode_Down Index， Button， Shift， 1， r， rr， rrr， FCP， LCP， X， Y
Exit Sub
   Case "Rightcurve"
   Pnode(Index).Visible = False： PolyCurveNode = True
 BezNode_Down Index， Button， Shift， 2， r， rr， rrr， FCP， LCP， X， Y
 Exit Sub
End Select
If PolyShape(B) = B Then 'set for line move
BezHandleMove Index
Toggle = True： P = 1： Pnode(Index).Visible = False
   X1 = Pnode(Index).Left + 45： Y1 = Pnode(Index).Top + 45
   X2 = X1： Y2 = Y1： Exit Sub
End If
   Case LineTool 'Set start of new line segment
      X1 = Pnode(Index).Left + 45： Y1 = Pnode(Index).Top + 45
      X2 = X1： Y2 = Y1
```

```
End Select
End Sub
```

Pnode 图象控制对象用于编辑多边形对象的线段。Pnode 可用来连接线段或作为 Bézier 曲线的控制点。

If CurrentTool=3：本程序采用了节点编辑的一个较小的版本。不允许向开或闭多边形中添加任何额外的线段，同时只能沿着一个方向构造多边形。例如，不可以先画一条线，然后从该线段另一端的 Pnode 节点引出新的线段。要做到这一点需要附加数组混序代码。

XX2=E(0):YY2=S(0)：这些数值是多边形的边界框尺寸，在移动线段之后可用来清除旧的图象区域。

下列变量告知例程事件哪一个 Pnode 连接着给定的多边形线段。多边形中每条线段都被赋以编号，每条线段的始末点都是根据其编号索引的。

```
r=FirstPolyPt(B)+Index 'Index+1
rr=FirstPolyPt(B)+Index-1 'Index
rrr=FirstPolyPt(B)+Index-2 'Index-1
FCP=FirstPolyPt(B) 'First object
LCP=LastPolyPt(B)-1 'Last object
```

为了节省内存，对每个高亮的多边形要重新载入 Pnode。Pnode 控制对象的索引号始于 0 并且终止于多边形中最后一个 Pnode。通过赋给 Pnode 的索引属性可以知道其编号；在绘制对象时，也可以从数组 FirstPolyPt 和 LastPolyPt 中获知始末多边形线段的编号。通过加或减这些已知数值就可以用 r,rr,rrr,FCP 或 LCP 等变量找到任何多边形线段的连接点。

可以把 r,rr,rrr,FCP 和 LCP 变量放入 Declaration（声明）中或 Module（模块）文件中并看作常数以节省空间。上述 5 个变量在整个 Pnode 事件及几个过程中被重复引用。

PIL and PIT(PnodeIndex.Left 和 PnodeIndex.Top)：每个 Pnode 的中心点。可以把 Pnode 的位置同多边形线段被索引的 X 和 Y 坐标作一比较。如果 Pnode 的中心点等于对应着 r,rr,rrr,FCP 和 LCP 的 X 和 Y 值，即得到一匹配。否则，r,rr,rrr,FCP 和 LCP 等五个索引值将置为 0，同时程序跳过该行代码。

If p1X(FCP)=PIL：每条多边形曲线都有一个起始点(p1.X 和 p1.Y)与一个终止点(p2.X 和 p2.Y)。如果 p1 点匹配，则把 Pnode 标记为 Leftcurve；如果 p2 点匹配，则把 Pnode 标记为 Rightcurve。如果 Pnode 连接着两条 Bézier 曲线，一条曲线的 P1 与另一条曲线的 P2 相接，则把 Pnode 标记为 2curves。

Select Case Pnode(Index).Tag：根据 Pnode 标记名，执行相应的 Case 块。

Case "Leftcurve","2curves"：如果 Pnode 具有这两个标记之一，则在 BezNode_Down 过程（清单 15.5）中会给 Pnode 赋以 Lever 参数值 1。Lever 值（第三个参数）表示曲线连接着图柄或节点的一侧。在后续的事件和过程中，Lever 的数值标志着曲线被移动的一侧，然后相应地放置连接着图柄和节点的虚线。

Case "Rightcurve"：把 BezNode_Down 的第 3 个参数改为 2，表示连接到 Bézier 曲线另一侧的图柄与节点上的第二条虚线。

If PolyShape(B)=B Then：被移动的 Pnode 连着一到两条直线但没有连着曲线。使用 Pnode_MouseMove 事件(清单 15.18)中独立的过程来移动这种折线连接。

BezHandleMove：擦除并重置虚线，同时重新放置相连的图柄。

Toggle=True:P=1：Toggle 变量在 Pnode_MouseMove 事件中用作标志，触发 Poly_LineMove 过程以重绘移动的线段。在每次移动前必须把 P 变量标志置为 1，此标志用来循环一个绘图例程。

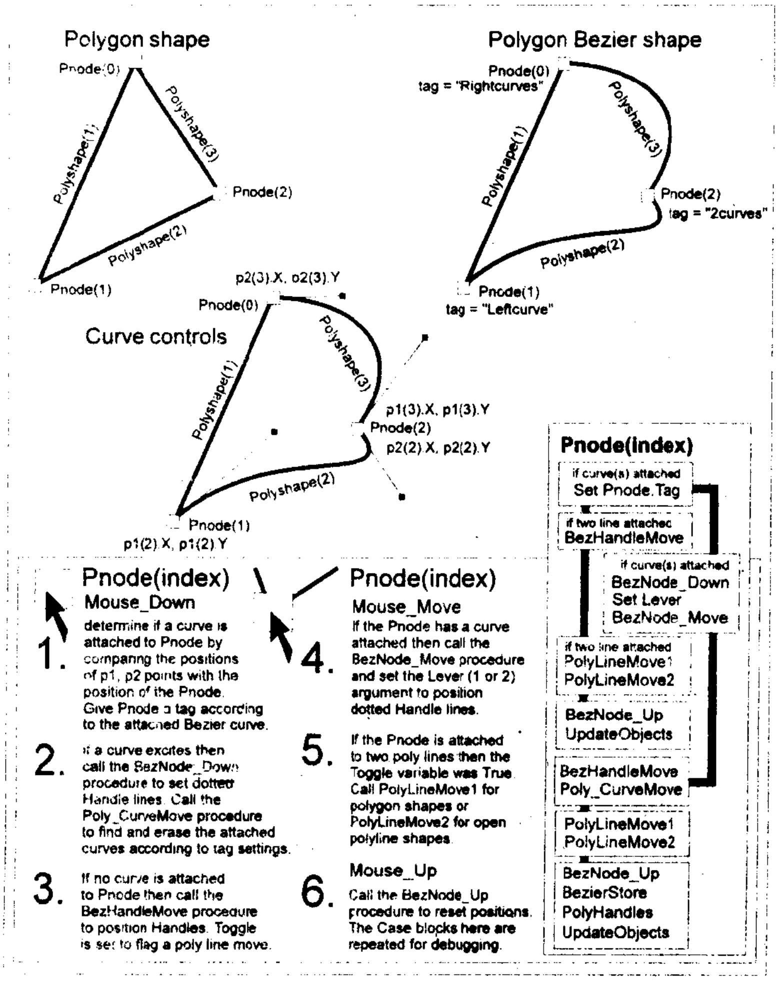
Pnode(Index).Visible：在移动时隐藏 Pnode，以免影响移动线段的绘制。

X1,Y1,X2,Y2：设置要绘制的线段的起始位置，以取得活动线的效果。

Case LineTool：如果选择此项工具，就不能移动 Pnode，但可以从 Pnode 引出一条新线段。这和 Join Nodes 程序(第 13 章)产生多边形对象的每条新线段的做法相类似。

X1,Y1,X2,Y2：设置将绘制的线段的起始位置，以取得活动线段的效果。

用几个过程和事件来计算和绘制连接着 Pnode 的多边形对象。图 15.12 说明了把一个 Pnode 移到新坐标时发生的一连串事件。清单 15.19 显示了如何引用每个过程调用。


Polygon shape
Pnode(0)
Polyshape(1)
Polyshape(3)
Pnode(2)
Polyshape(2)
Pnode(1)
Polygon Bezier shape
Pnode(0)
tag = "Rightcurves"
Polyshape(1)
Polyshape(3)
Pnode(2)
tag = "2curves"
Polyshape(2)
Pnode(1)
tag = "Leftcurve"
Curve controls
p2(3).X, p2(3).Y
Pnode(0)
Polyshape(1)
Polyshape(3)
p1(3).X, p1(3).Y
Pnode(2)
p2(2).X, p2(2).Y
Polyshape(2)
Pnode(1)
p1(2).X, p1(2).Y
Pnode(index)
Mouse_Down
Pnode(index)
Mouse_Move
Mouse_Up
Pnode(index)
if curve(s) attached
Set Pnode.Tag
if two line attached
BezHandleMove
if curve(s) attached
BezNode_Down
Set Lever
BezNode_Move
if two line attached
PolyLineMove1
PolyLineMove2
BezNode_Up
UpdateObjects
BezHandleMove
Poly_CurveMove
PolyLineMove1
PolyLineMove2
BezNode_Up
BezierStore
PolyHandles
UpdateObjects

1. determine if a curve is attached to Pnode by comparing the positions of p1, p2 points with the position of the Pnode. Give Pnode a tag according to the attached Bezier curve.
2. if a curve excites then call the BezNode_Down procedure to set dotted Handle lines. Call the Poly_CurveMove procedure to find and erase the attached curves according to tag settings.
3. If no curve is attached to Pnode then call the BezHandleMove procedure to position Handles. Toggle is set to flag a poly line move.
4. If the Pnode has a curve attached then call the BezNode_Move procedure and set the Lever (1 or 2) argument to position dotted Handle lines.
5. If the Pnode is attached to two poly lines then the Toggle variable was True. Call PolyLineMove1 for polygon shapes or PolyLineMove2 for open polyline shapes.
6. Call the BezNode_Up procedure to reset positions. The Case blocks here are repeated for debugging.

图 15.12　Pnode 的过程链

清单 15.19 移动多边线段到新位置

```
Sub Pnode_MouseMove (Index%，Button%，Shift%，X As Single，Y As Single)
PL = (X + Pnode(Index). Left)：PT = (Y + Pnode(Index). Top)
Xstr = Format $ (PL \ PixelX，"000")：Ystr = Format $ (PT \ PixelY，"000")
XYLabel. Caption = "[" + Xstr + "，" + Ystr + "]"
If CurrentTool = 3 And Index < NodeTotal - 1 Then Exit Sub
If CurrentTool = 3 And Index < LastPolyPt(B) Then Exit Sub
''''''''''''''''''''''''''''''''''''''''''''''''''''
r = FirstPolyPt(B) + Index      'Index +1
rr = FirstPolyPt(B) + Index - 1      'Index
rrr = FirstPolyPt(B) + Index - 2 'Index -1
FCP = FirstPolyPt(B)      'First object
LCP = LastPolyPt(B) - 1      'Last object
''''''''''''''''''''''''''''''''''''''''''''''''''
If Button Then
If CurrentTool <> 1 And Index < NodeTotal - 1 Then Exit Sub
Select Case CurrentTool
    Case BézierTool
If PolyCurveNode = True Then
Select Case Pnode(Index). Tag
    Case "Leftcurve"，"2curves"
BezNode_Move Index，Button，Shift，1，r，rr，rrr，FCP，LCP，X，Y：Exit Sub
    Case "Rightcurve"
BezNode_Move Index，Button，Shift，2，r，rr，rrr，FCP，LCP，X，Y：Exit Sub
End Select
End If
If Toggle = True And FlagPolyLine(FCP) = FirstPolyPt(B)
        Then Poly_LineMove2 Index，Button，Shift，X，Y
Exit Sub
If Toggle = True And FlagPoly(FCP) = FirstPolyPt(B)
        Then Poly_LineMove1 Index，Button，Shift，X，Y
Exit Sub
    Case LineTool 'Draw new Line segment
        ArtWork. Line (X1，Y1)-(X2，Y2)
            X2 = X + Pnode(Index). Left：Y2 = Y + Pnode(Index). Top
        ArtWork. Line (X1，Y1)-(X2，Y2)
End Select
End If
End Sub
```

PL,PT：这两个变量给出每个 Pnode 控制对象的当前 X 和 Y 坐标。

If CurrentTool = 3：本程序不能在已设置好的多边形上增加折线，也不能从多边形线段的相对端引出线段。要满足以上要求，就需要添加打乱数组下标的例程来适应绘图中新的 Pnode 路径，亦即需要另添代码来打断多边形的路径或将之分隔为几个多边对象。

r,rr,rrr,FCP,LCP：Pnode_MouseDown 事件中有关于这些变量的描述。

If Button Then：只有当鼠标按钮被按下时，才可继续此过程。

If CurrentTool <>1：如果选择了错误的工具则停止绘图。

Select Case CurrentTool：根据所选择的工具，执行相应的 Case 块。

Case BézierTool：使用 **Bézier**(形状)工具来移动一个多边形线段。

If PolyCurveNode=True Then：如果 Pnode 连接着一条或更多条 Bézier 曲线，Pnode_MouseDown 过程置此标志为 True。

Select Case Pnode(Index). Tag：如果 Pnode 被标记为 Leftcurve 或 Rightcurve，必须在 BezNode_Move 过程(清单 15.6)中指定 Lever 参数为 1。此 Lever 值(第 3 个参数)表示图柄或节点相连的一侧曲线。在后续事件和过程中，此 Lever 值标志着要移动曲线的哪一侧。这样即可相应地放置连接图柄和节点的虚线。

Case "Rightcurve"：将 BezNode_Move 的第三个参数置为 2。此值表示连接着 Bézier 曲线另一侧的图柄和节点的第二条虚线(控制线)。

If Toggle=True：Flag PolyLine 标志表示对象为开多边形。当首末点没有连接时即为开多边形(如字符 W)。

If Toggle=True：FlagPoly 变量标志着对象为闭多边形。当首末点相接时即为闭多边形(如三角形)。

可使用 Poly_LineMove2(见 ARTAPI1.FRM 源代码)和 PolyLineMove1 过程来绘制开或闭多边形的活动线。除去禁止在开多边形上绘制闭合线段的几行代码外，这两个过程彼此很类似。也可以把这两个过程合并为一体。

Poly_LineMove2：绘制开多边形的线段。

Poly_LineMove1：绘制闭多边形的线段。

Case LineTool：如果选择画线工具，执行后续代码块。

可使用 LineTool 工具从当前 Pnode 控制对象开始画新的线段。

ArtWork. Line：绘制新线段。

X2,Y2：交换始末点以取得移动效果。

ArtWork. Line：重绘线段以擦除旧图象。

在移动 Pnode 到新位置后，使用清单 15.20 中的 Pnode_MouseUp 事件作全面整理。将调用 BezNode_Up 过程来保存所有新坐标点，然后删去 Pnode 的标记名，并将变量标志置回初始值。

清单 15.20　设置已移动多边形的新坐标

```
Sub Pnode_MouseUp (Index%, Button%, Shift%, X As Single, Y As Single)
If CurrentTool = 3 And Index < NodeTotal - 1 Then Exit Sub
If CurrentTool = 3 And Index < LastPolyPt(B) Then Exit Sub
'''''''''''''''''''''''''''''''''''''''''''''

r = FirstPolyPt(B) + Index                'Index +1
rr = FirstPolyPt(B) + Index - 1      'Index
rrr = FirstPolyPt(B) + Index - 2 'Index -1
FCP = FirstPolyPt(B)                          'First object
LCP = LastPolyPt(B) - 1                      'Last object
'''''''''''''''''''''''''''''''''''''''''''''

Select Case CurrentTool
    Case BézierTool
If PolyCurveNode = True Then
```

```
        Select Case Pnode(Index).Tag
         Case "Leftcurve"、"2curves"
              BezNode_Up Index, Button, Shift, 1, X, Y
              PolyCurveNode = False: Pnode(Index).Visible = True: Pnode(Index).Tag = ""
                  Exit Sub
         Case "Rightcurve"
              BezNode_Up Index, Button, Shift, 1, X, Y
              PolyCurveNode = False: Pnode(Index).Visible = True
              Pnode(Index).Tag = "": Exit Sub
    End Select
    End If
    If Toggle = True Then Poly_Update Index, Button, Shift, X, Y: Exit Sub
        Case LineTool 'set new line segments
            Poly_gon Index
    End Select
    End Sub
```

If CurrentTool＝3：本程序没有设置为在已构成的多边形上绘制多余折线，也不能从多边形的相对端引出折线。

r，rr，rrr，FCP，LCP：Pnode_MouseDown 事件中有关于这些变量的描述。

Select Case CurrentTool：根据所选工具，执行相应 Case 块。

Case BézierTool：使用 **Bézier**(形状)工具。

If PolyCurveMove＝True Then：如果 Pnode 连接一条或多条 Bézier 曲线，由 Pnode_MouseDown 过程将此标志置为 True。

Select Case Pnode(Index). Tag：如果 Pnode 具有某个 Select Case Tag 标记，则需指定 BezNode_Up 过程(清单 15.7)的 Lever 参数为 1。此 Lever 值(第 3 个参数)表示图柄和节点相连的一侧曲线。在后续的事件和过程中，此 Lever 值标志着要移动曲线的哪一侧，这样即可相应地放置连接图柄和节点的虚线。

Case "Rightcurve"：设置第三个 BezNode_Up 参数值为 2，此值表示连着 Bézier 曲线另一侧的图柄和节点的虚线。

If Toggle＝True：Pnode 连接的是折线，而不是曲线。Poly_Update 过程将重置当前 Pnode 的 X 和 Y 坐标和其他数值。

Case LineTool：使用画线工具来加入新线段。

Poly_gon Index：一旦构成多边对象，此过程即加入 Pnode。如果最后一条折线段已靠近第一条线段，这两条线段会连到一起以构成多边形。

注释： 在 ArtAPI 程序中有 12 个多边形过程。大部分过程都可以用 If/Then 或 Choose 语句把相似过程分组归并起来。例如 Poly_CurveMove1 和 Poly_CurveMove2，还有 PolyLineMove1 和 PolyLineMove1 过程均可合并为一个过程。分别讲述这些过程是为了说明每个折线和多边形绘制例程。

Poly_gon，Poly_Line 和 Poly_Update 过程用于绘制对象，而 Poly_Node 和 Poly_Reload 用于定位多边形中的 Pnode。

下表列出了从 Pnode 图象控制对象分支出去的相关过程：

移动连接曲线和多边形的 Pnode：Poly_CurveMove1。移动连接曲线和折线的 Pnode：

Poly_CurveMove2。移动连接直线和多边形的 Pnode：Poly_LineMove1。移动连接直线和折线的 Pnode：Poly_LineMove2（参见 ARTAPI1.FRM 的源代码）。

绘制已完成的多边形：Poly_gon（清单 15.24），Poly_Lines（清单 15.25），Poly_Curves（清单 15.23）以及 Poly_Update（清单 15.29）。

向多边形中添加或重载 Pnode：Poly_Nodes（清单 15.27）和 Poly_Reload（清单 15.28）。

计算多边形的边界框以及调整大小特性：Poly_Handles（清单 15.13）和 Poly_Stretch（清单 15.30）。

15.27　编辑多边形中的曲线

清单 15.21 中的 Poly_CurveMove1 过程在移动线段时绘制与 Pnode 相连的曲线段。本例程用于移动闭合多边形对象并被分为三个主要部分，这样更便于理解 Poly_CurveMove1 过程是如何工作的。

清单 15.21　绘制与闭合多边形相连的可移动曲线

```
Sub Poly_CurveMove1 (Index%, Button%, Shift%, Lever%, X!, Y!)
r = FirstPolyPt(B) + Index               'Index +1
rr = FirstPolyPt(B) + Index - 1          'Index
rrr = FirstPolyPt(B) + Index - 2         'Index -1
FCP = FirstPolyPt(B)                          'First object
LCP = LastPolyPt(B) - 1                        'Last object
LastIndex = (LastPolyPt(B) - FirstPolyPt(B)) - 1
'''''''''''''''''''''''''''''''''''''''''''''''''''''
Select Case Pnode(Index).Tag
    Case "Leftcurve"
If Index = 0 Then BB = FCP: ii = LastIndex: PT = FCP: Z = 1
If Index > 0 Then BB = rrr: ii = Index - 1: PT = r
    Case "Rightcurve"
If Index = 0 Then BB = LCP: ii = Index + 1: PT = LCP
If Index > 0 Then BB = rr: ii = Index + 1: PT = rr
If Index = LastIndex Then BB = rr: ii = 0: PT = rr
    Case "2curves"
If Index = 0 Then BB = LCP: PT = LCP: Lever = 2
If Index > 0 Then BB = rr: PT = rr: Lever = 2
End Select
```

注释：　在对 PolyCurveMove1 过程的第一分部进行说明后，将列出后续代码清单。

下列变量可使事件识别哪个 Pnode 连接着给定多边形。多边形中每个多边对象都被赋以一个编号，每个多边形的始末点均以此编号为索引。

r=FirstPolyPt(B)+Index 'Index+1
rr=FirstPolyPt(B)+Index-1 'Index
rrr=FirstPolyPt(B)+Index-2 'Index-1
FCP=FirstPolyPt(B) 'First object
LCP=LastPolyPt(B)-1 'last object

Select Case Pnode(Index).Tag：取决于哪一侧曲线连接着 Pnode。

Case "Leftcurve":"Rightcurve":"2curves"：如果存在 Bézier 曲线，则 BB 的值表示该曲线起始点的索引号。

如果存在线段，则 ii 的值表示该线段起始点的索引号。

PT 的值表示 4 个 Bézier 曲线控制点的索引号。

当 Pnode 连着两条曲线时(2curves)，两个 Lever 都用来绘制曲线。

清单 15.21　绘制可移动曲线(续)

```
If CountPolyPts = 1 Then 'calculate the number of PolyPts
    ArtWork.DrawMode = 6: ArtWork.DrawStyle = 0
If Z = 1 Then fp = 0 Else fp = 1
    API = Polyline(ArtWork.hDC, Npt(fp), PolyPts)'draw shape first
End If
i = 1
''''''''''''''''''''''
For DrawShape = 1 To 2
If BB = FlagBezier(B + NC) And BB = PolyShape(B + NC) Then
Select Case Lever
    Case 1 'BezNode1
   p1.X = X + Pnode(Index).Left: p1.Y = Y + Pnode(Index).Top
   p2.X = p2X(PT): p2.Y = p2Y(PT)
   h1.X = h1X(PT): h1.Y = h1Y(PT)
   h2.X = h2X(PT): h2.Y = h2Y(PT)
   Poly_curves i
    Case 2 'BezNode2
   p2.X = X + Pnode(Index).Left: p2.Y = Y + Pnode(Index).Top
   p1.X = p1X(PT): p1.Y = p1Y(PT)
   h1.X = h1X(PT): h1.Y = h1Y(PT)
   h2.X = h2X(PT): h2.Y = h2Y(PT)
   Poly_curves i
End Select
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Else 'Draw a line
    Npt(i).X = (Pnode(ii).Left + 45) \ PixelX
    Npt(i).Y = (Pnode(ii).Top + 45) \ PixelY: i = i + 1
End If
''''''''''
If Z = 1 Then
    Npt(0).X = (Pnode(LastIndex).Left + 45) \ PixelX
    Npt(0).Y = (Pnode(LastIndex).Top + 45) \ PixelY: i = i - 1
End If
BB = BB + 2
If Pnode(Index).Tag = "2curves" Then Lever = 1:
              BB = BB - 1: NC = 1: PT = PT + 1
If Pnode(Index).Tag = "2curves" And Index = 0 Then
    Lever = 1: BB = FCP: NC = -LastIndex: PT = FCP
End If
Next DrawShape
```

注意： 在下列文字之后继续列出代码清单。

清单 15.21 的第二部分计算 PolyPts 的数目并传给 API 函数 Polyline，这样此过程将只绘出移动的两条线段而非整个多边对象。

只应绘制与正在移动的 Pnode 相连的两条线段。首先，从存放所有点的数组中计算出索引点数目。

下列语句描述了清单 15.21 中 Poly_CurveMove1 过程的第二部分：

If CountPolyPts＝1 Then：在每次移动前 CountPolyPts 变量等于 0。在观察与此 Pnode 相连的每条线段之前并不知道 API 函数 Polyline 将用多少 PolyPts。本过程在第 2 次鼠标移动调用到来之前跳过 If/Then 例程块。

API＝Polyline：在计算 PolyPts 后绘制头两条线段。

下一组语句计算 PolyPts 数目并加入 Npt 数组中。

For DrawShape＝1 to 2：循环两次可以找到与 Pnode 控制对象相连的两条线段。

If BB ＝ FlagBézier(B＋NC)：如果第一条线段是一条 Bézier 曲线，则获取每个正确的曲线点并存入 Npt 数组。B 变量相当于与 Pnode 相连的第一条曲线。如果 Bézier 曲线是 Pnode 相连的第 2 条曲线，则在下次循环中 NC 将等于 1。

Select Case Lever：Pnode 事件设置 Lever(1 或 2)。记住一条 Bézier 曲线是从中央分开的并且在每一端都被图柄和节点控制(p1，h1 和 p2，h2)。

Case 1：类似于第 3 章中 Artisan 程序的 BezNode1 事件所用的例程。p1.X 和 p1.Y 控制沿着鼠标的拖动轨迹移动。其余三个 Bézier 控制点从所存放的数组中取出并被传给 Poly_curves 过程，后者使用"deCasteljau"分割公式得到 Bézier 曲线。

Poly_curves I：在 Poly_curves 过程中所有曲线点均已保存在 Npt 数组中。参数 I 记录着用于 API 折线 Npt 数组的每一个数组索引位置。

Case 2：类似于 Artisan 程序中(第 3 章)BezNode2 事件所用的例程。除了使用曲线的左侧坐标外，与 Case 1 基本相同。

Else：如果折线段不是曲线，则必为直线。本过程将执行行后续 Else/Then 语句块。

Npt(i).X＝(Pnode(ii).Left＋45)：从当前 Pnode(i)到下一个 Pnode(ii)画一条线。

BB＝BB＋2：如果是直线与曲线相连，BB 将在下次循环中等于 FlagBézier，这将会处理连接的 Bézier 曲线。

If Pnode(Index).Tag＝"2curves" Then：如果 Pnode 连着两条曲线，则需绘制另一条曲线。这样就修改了所有指向第一条曲线的当前数值以指向第二条曲线。

If Pnode(Index).Tag＝"2curves" And Index＝0 Then：多边形的第一个 Pnode 是一个特殊情形。必须改动所有数值以寻找多边形中最后一条曲线段。

Next DrawShape：返回 For DrawShape 语句并为与 Pnode 控制对象相连的第 2 条折线段计算 PolyPts。

清单 15.21　(续)

```
PolyPts = i - 1
If CountPolyPts = 0 Then
    CountPolyPts = 1
    ArtWork.AutoRedraw = True
ArtWork.DrawMode = 16: ArtWork.DrawWidth = 4: ArtWork.DrawStyle = 0
If Z = 1 Then fp = 0 Else fp = 1
    ArtWork.AutoRedraw = False
    API = Polyline(ArtWork.hDC, Npt(fp), PolyPts)
    ArtWork.DrawMode = 6: ArtWork.DrawWidth = 1
End If
If Z = 1 Then fp = 0 Else fp = 1
    API = Polyline(ArtWork.hDC, Npt(fp), PolyPts) 'redraw shape
End Sub
```

Poly_CurveMove1 过程的最后部分擦除屏幕上旧的多边形图象。实际上它只在移动开始时清除一次，然后跳向最后两行代码来绘制被移动的两条多边形线段。

PolyPts=i－1：把 PolyPts 的数目传给 API 函数 Polyline 以绘出两条折线段。

If CounterPolyPts=0 Then：这是头一次 MouseMove 事件，所以要擦除旧的多边形图象。

CountPolyPts=1：使 CountPolyPts 等于 1，这样就不会再调用这块图象擦除语句。对每次新的鼠标移动，本过程开头的 API Polyline 语句将执行，这样在 Inverse（反转）绘制模式下将可执行两次绘图调用以擦去及绘制多边形。第二次 API Polyline 调用在本过程的末尾。

ArtWork.AutoRedraw=True：必须从内存位图中擦去图象。改变 AutoRedraw 属性是个不错的技巧，这样不会减慢程序的速度。首先把 DrawMode 属性置为 16（白色），然后在绘制前把 AutoRedraw 设回 False。当内存位图获取此信息后，就可以在过程之前的一个 MouseMove 间隔中更新图象。如果在 AutoRedraw=True 和 AutoRedraw=False 之间加入一条图形语句，图象就会立刻刷新。这将冻结所有事件直到处理完图象并会使下一幅图象快速显示出来。

IF Z=1 Then fp=0 Else fp=1：连接第一条和最后一条线段的 Pnode(0)是一个特别情形。Polyline 函数绘出的是一条连续线，所以需要反转 Npt 的第一个索引以首先绘制最后一条折线段。fp 变量是 First Point（第一点）的缩写。

API=Polyline：用白色绘制多边形以擦除旧图象。

If Z=1 Then fp=0 Else fp=1：fp 变量是 First Point 的缩写，是本过程用来初始化多边形的第一个绘制点。

API=Polyline：最后一个 API 图形函数，以 Inverse 模式不断地重绘线段。

清单 15.22 中的 Poly_CurveMove2 过程同 Poly_CurveMove1 比较类似，故只列出了不同的部分。

清单 15.22 绘制开多边形中的可移动曲线

```
Sub Poly_CurveMove2(Index%,Button%,Shift%,Lever%,X!,Y!)
'Poly_CurveMove2 use a routine similar to Poly_CurveMove1 (listing 15.2):
'Only the code lines which are different are listed in the preceding:
Else 'Draw a line
Select Case Index
    Case 0, LastIndex
```

```
        Case Else
            Npt(i).X = (Pnode(ii).Left + 45) \ PixelX
            Npt(i).Y = (Pnode(ii).Top + 45) \ PixelY: i = i + 1
        End Select
    End If
    BB = BB + 2
    If Pnode(Index).Tag = "2curves" Then Lever = 1: BB = BB - 1:
                            NC = 1: PT = PT + 1
    If Pnode(Index).Tag = "2curves" And Index = 0 Then
        Lever = 1: BB = FCP: NC = -LastIndex: PT = FCP
    End If
    Next DrawShape
    '''''''''''''''
    PolyPts = i - 1
    If CountPolyPts = 0 Then
        CountPolyPts = 1
        ArtWork.AutoRedraw = True
        ArtWork.DrawMode = 16: ArtWork.DrawWidth = 4: ArtWork.DrawStyle = 0
        ArtWork.AutoRedraw = False
        API = Polyline(ArtWork.hDC, Npt(1), PolyPts)
        ArtWork.DrawMode = 6: ArtWork.DrawWidth = 1
    End If
        API = Polyline(ArtWork.hDC, Npt(1), PolyPts) 'redraw shape
    End Sub
```

Poly_CurveMove2 与 Poly_CurveMove1 大体相同。不同的是使用 Pnode(0)或最后一个节点 Pnode(LastIndex)绘制第一条和最后一条多边形段的方式。

同样，在此也必须修改三个 API 函数 Polyline 函数以使用 Npt 数组的第一个元素开始绘图。这里删去了 Poly_CurveMove1 过程中的 If Z=1 Then fp=0 Else fp=1 语句。

15.28 绘制多边形的每一条曲线

这里略去了 Poly_Curves 过程的“deCasteljau”分割公式以节省空间。本公式可参见第3章中的“BézierCurves(Bézier 曲线)”部分。

“deCasteljau”算法用 33 个曲线点来组成一条 Bézier 曲线。Poly curves 过程从数组 C1(1 to 33)中获取这 33 个点，然后在 Artwork 图片框、ArtMask 图片框或打印机上绘制 Bézier 曲线。每个设备都使用了一种完全不同的内部标度，故每个设备都有独立的绘制例程。

如果使用 Windows 32 位操作系统，就可以用 GUI32 API PolyBézier 和 PolyBézierTo 函数来替代清单 15.23 的大部分代码。

清单 15.23 用于 Bézier 曲线的 deCasteljau 分割算法

```
Sub Poly_curves (i As Integer)
' the dividing algorithm has been omitted to save space
' refer to the source code or Chapter 2—Bézier curves.
If FlagPrinter = True Then GoTo Printcurve
If FlagArtMask = True Then GoTo ArtMaskcurve
Npt(i).X = p1.X \ PixelX: Npt(i).Y = p1.Y \ PixelY
    For p = 2 To 32
        Npt(p + i - 1).X = C1(p).X \ PixelX
        Npt(p + I - 1).Y = C1(p).Y \ PixelY
    Next p
Npt(33 + I - 1).X = p2.X \ PixelX
Npt(33 + I - 1).Y = p2.Y \ PixelY
I = I + 33
Exit Sub
''''''''''''''''''''''''''''''''''''''''''

Printcurve:
Dim RightM As Single
Dim BottomM As Single
LeftM = PageX1: TopM = PageY1 'Adjust Margins
APIH = GetDeviceCaps(Printer.hDC, 8)
APIV = GetDeviceCaps(Printer.hDC, 10)
HPrinterDPI! = APIH / (PageX2 - PageX1)
VPrinterDPI! = APIV / (PageY2 - PageY1)
RightM = HPrinterDPI!        'Adjust Scale size
BottomM = VPrinterDPI!
Npt(I).X = (p1.X - LeftM) * RightM: Npt(I).Y = (p1.Y - TopM) * BottomM
    For p = 2 To 32
        Npt(p + I - 1).X = (C1(p).X - LeftM) * RightM
        Npt(p + I - 1).Y = (C1(p).Y - TopM) * BottomM
    Next p
Npt(33 + I - 1).X = (p2.X - LeftM) * RightM
Npt(33 + I - 1).Y = (p2.Y - TopM) * BottomM
I = I + 33
Exit Sub
''''''''''''''''''''''''''''''''''''''''''

ArtMaskcurve:
Npt(I).X = (p1.X - ArtMask.Left) \ PixelX
Npt(I).Y = (p1.Y - ArtMask.Top) \ PixelY
    For p = 2 To 32
        Npt(p + I - 1).X = (C1(p).X - ArtMask.Left) \ PixelX
        Npt(p + I - 1).Y = (C1(p).Y - ArtMask.Top) \ PixelY
        Next p
Npt(33 + I - 1).X = (p2.X - ArtMask.Left) \ PixelX
Npt(33 + I - 1).Y = (p2.Y - ArtMask.Top) \ PixelY
I = I + 33
Exit Sub
End Sub
```

If Flagprinter:在打印机页上绘制曲线。

If FlagArtMask:使用鼠标触及了曲线段。

曲线的绘制点存放于数组C1,然后赋给Npt数组,这样Polyline或Polygon函数即可绘制曲线。第3章详细讲述了如何标绘曲线点。

需要往三个设备上绘制Bézier曲线。设备可以是两个Visual Basic图片框之一(Artwork和ArMask)或打印机。

Artwork图片框控制对象代表桌面。向此设备绘图需要将X,Y坐标从twip单位转换为象素单位。

另一个图片框为ArtMask,此图片框是隐藏的,用作多边形中单个对象或线段的内存位图图象。每次用鼠标触及多边形中的对象或线段时都要重新调整ArtMask控制对象的尺

寸。使用 ArtMask 图片框的目的是为了找出对象的轮廓以便选择或移动多边形。每次重新调整 ArtMask 图片框控制对象的大小后，内部 twip 刻度总是以左上角为 0,0 坐标点。为了绘制曲线，需要从相应的曲线标绘点坐标 X，Y 中减去 ArtMask 图片框的 Left 和 Top 位置值。

另一个绘图设备是打印机。各厂家的打印机内部页面尺寸互不相同，所以必须调整数组以配合所用的型号。可用第 14 章的 APIprint 程序来测试手工及 API 打印方法与函数。

为了打印曲线，要调整页面边界误差。Artwork 的桌面页位于该控制 Left 和 Top 属性值的中央。打印机页则起始于 0,0 坐标。然后，需要乘一个比例因子来压缩打印机内部点每英寸(DPI)定标模式。

下列 Poly_Curves 过程中的语句正确地在打印机页上绘制 Bézier 曲线。

APIH＝GetDeviceCaps(Printer. hDC,8)：取得打印机水平分辨率(象素单位)。

APIV＝GetDeviceCaps(Printer. hDC,10)：取得打印机垂直分辨率(象素单位)。

HprinterDPI! ＝API H/(PageX2－PageX1)：计算打印机页宽和 Artwork 图片框控制对象中要打印区域水平宽度之间的误差。PageX1 和 PageX2 表示在 UpdateDesktop 过程中绘制在桌面上的页面图象的宽度。

VPrinterDPI! ＝APIV/(PageY2－PageY1)：计算打印机页面高度和 Artwork 图片框控制对象中要打印区域的垂直高度之间的误差。PageY1 和 PageY2 表示在 UpdateDesktop 过程中绘制在桌面上的页面图象的高度。

RightM＝HPrinterDPI! 'Adjust Scale size：按百分比加减这些数值以调节要打印的对象。

BottomM＝VPrinterDPI!：按百分比加减这些数值以调节要打印的对象。

15.29　封闭一个多边形

清单 15.24 中的 Poly_gon 过程向正在绘制的线段添加额外节点。如果正拖动的线段进入了多边形的第一个 Pnode 的 8 象素宽的区域，两个 Pnode 将合并以形成一个多边形。

清单 15.24　向新的多边形线段加入 Pnode 或封闭一个多边形

```
Sub Poly_gon (Index)
If Counter = 1 Then FlagLine(B) = 0: ZZ = B: PolyShape(B) = ZZ:
              FirstPolyPt(B) = ZZ:Snap = 120
If X2 > Node(0).Left - Snap And X2 < Node(0).Left + 90 + Snap And
              Y2 > Node(0).Top - Snap And Y2 < Node(0).Top + 90 + Snap Then
  Total = Total + 1: B = Total
  DrawOrder(B) = Total
  FlagLine(B) = 0: PolyShape(B) = B: FirstPolyPt(B) = ZZ: FlagPoly(ZZ) = ZZ
  W(B) = X1: E(B) = X2: N(B) = Y1: S(B) = Y2
  LineWidth(B) = ArtWork.DrawWidth
    E(B) = Node(0).Left + 45: S(B) = Node(0).Top + 45
    NodeTotal = 2: PolyClose = False
    For r = FirstPolyPt(B) To B + 1
        LastPolyPt(r) = B + 1
    Next r
```

```
        FillValue(B) = 1
        UpdateObjects
        Exit Sub
    Else
        PolyClose = True
        Poly_Nodes Index 'Load new nodes
    End If
    End Sub
```

If Counter=1:Counter 代表被加入对象的每个 Pnode。绘制对象时需要把第一个节点的坐标传给 API 函数 Polyline 作为绘图起点。

FlagLine(B)=0:原来的直线对象已成为折线段,故清除索引标志。

ZZ=B:把第一个折线段编号临时存放在变量 ZZ 中。每添加一条新线段,ZZ 都随着增加,直到最后一条线段与第一条线段连接起来形成了一个多边形。

PolyShape(B)=ZZ:原来的直线对象现已是多边形对象,故保存索引。

FlagPoly=ZZ:保存正绘制的对象类型,这样 UpdateObject 过程(清单 15.33)就知道进行怎样的绘图操作。

FirstPolyPt(B)=ZZ:标记第一条线段(第一个 Pnode),这样即可找到多边形上介于第一个和最后一个 Pnode 之间的所有点。

Snap=120:通过声明两个节点可以合并的最大距离来使最后一条线段的节点跳向头一条线段的节点。目前采用在所有方向上均为 120 twip,即 8 象素。可修改此数值以适合你的要求。

If X2>Node(0).Left-Snap And X2<Node(0).Left+90+Snap And 等等,:计算欲合并的两个节点间距离的公式。

Total=Total+1:B=Total:加入一新线段,故对象总数加 1。

FlagLine(B)=0:在合并为多边形后,设置最后一条线段的所有数值。

W(B)=X1:保存线段的边界框位置。

LineWidth(B)=ArtWork.DrawWidth:保存当前线宽。

E(B)=Node(0).Left+45:S(B)=Node(0).Top+45:E(B),S(B)相当于最后一条线段的终点坐标。现将它连向第一条线段的第一个节点。

NodeTotal=2:将节点数重置为默认值 2。

PolyClose:此标志告知程序多边形已完成。

For R=FirstPolyPt(B) To B+1:给每条多边形线段赋以多边形中最后一个多边形点的索引号,这样,就可以确定任何形状的最后点。

LastPolyPts(r)=B+1:本语句反映了此形状中最后一点,故可将之传给 API 函数 Polygon。

Fillvalue(B)=1:在填充颜色之前形状是透明的。当多边形在 UpdateObject 过程中被重绘时,Fillvalue 标志使 FillStyle 置为 1(透明)。

UpdateObjects:重绘显示在桌面上的所有对象。

Else:仍在向多边形添加线段,故多边形不闭合。

PolyClose:对象仍为开多边形,故保持此标志为 True 设置。

Poly_Nodes：仍在绘制过程中，故为新线段加入 Pnode。

15.30　绘制折线形状

Poly_Lines 过程设置了绘制开多边形所需的最终数值。如果用户在桌面击下鼠标或选择了另一个程序控制对象（比如滚动条或调色板），则 Artwork_MouseDown（清单 15.3）或 Artwork Lost_Focus 事件（见 ARTAPI.FRM 源代码）将执行此过程以绘制多边形。一个多边对象总是始于开多边形。如果将最后一个 Pnode 与第一个相接，对象自动成为闭合多边形。如果在绘制开多边形线段的过程中将焦点切换至其他任务，清单 15.25 中的 Poly_Lines 过程将卸去当前 Pnode 并将形状设置为当前的折线形式。

清单 15.25　设置一开折线形并卸去 Pnode

```
Sub Poly_Lines ()
    FlagPolyLine(ZZ) = ZZ
    On Error GoTo No_Node:
For r = 1 To NodeTotal - 1
    If NodeTotal > 0 Then Unload Pnode(r)
Next r
No_Node: Resume Next
NodeTotal = 2
    For r = FirstPolyPt(B) To B + 1
        LastPolyPt(r) = B + 1
    Next r
    FillValue(B) = 1
    UpdateObjects
End Sub
```

FlagPolyLine(ZZ)=ZZ：对象缺省为一个开折线形。ZZ 变量记录着第一条线段的指定编号，并被加入 FlagPolyLine 索引，这样，UpdateObjects（清单 15.33）即可使用 Polyline 绘图例程绘制形状。

On Error GoTo No_Node：卸载和载入节点有时会搞混。为了避免出现"索引已存在"或"索引不存在"等错误信息，在发生错误时跳过 Unload 语句。

For r=1 To NodeTotal-1：从 Pnode(1)直到最后一个 Pnode 索引，执行后续 For/Next 语句块。

警告：*决不要卸去初始 Pnode 控制对象，即 Pnode(0)。窗体中必须总保持一个这样的图象控制对象以复制(Load)此控制对象。*

If NodeTotal>0 Then Unload Pnode(r)：从折线形中卸去 Pnode。

No_Node:Resume Next：如果过程中发生错误，则跳至本行。

NodeTotal=2：在开始一新线段时将节点总数重置为缺省值。

For r=FirstPolyPt(B) To B+1：从第一条线段到最后一条，执行后续 For/Next 语句块。

LastPolyPt(r)=B+1：把最后一个线段点的索引赋给多边形中每条线段。

FillValue(B)=1：以透明方式即不采用颜色来填充每条线段。

UpdateObjects：重绘桌面上所有对象。

15.31 编辑折线形状

可使用清单 15.26 中的 Poly_LineMove1 过程两条连接着封闭多边形中 Pnode 控制对象的线段。Poly_LineMove2 过程(见 ARTAPI.FRM 源码)较为类似但用于移动开多边形中的两条线段。可以把这两个过程合并为一个过程。

清单 15.26 移动两条折线段到新位置

```
Sub Poly_LineMove1 (Index%, Button%, Shift As Integer, X As Single, Y As Single)
If Button Then
        LastIndex = (LastPolyPt(B) - FirstPolyPt(B)) - 1
        r = FirstPolyPt(B) + Index - 1
        rr = FirstPolyPt(B) + Index + 1
        If rr > LastIndex Then rr = FirstPolyPt(B) ' Last node point
    If Index = 0 Then r = FirstPolyPt(B) + 1: rr = LastPolyPt(B) - 1

Select Case Index
    Case 0
X1 = Pnode(LastIndex).Left + 45: Y1 = Pnode(LastIndex).Top + 45
X3 = Pnode(Index + 1).Left + 45: Y3 = Pnode(Index + 1).Top + 45
    Case LastIndex
X1 = Pnode(Index - 1).Left + 45: Y1 = Pnode(Index - 1).Top + 45
X3 = Pnode(0).Left + 45: Y3 = Pnode(0).Top + 45
    Case Else
X1 = Pnode(Index - 1).Left + 45: Y1 = Pnode(Index - 1).Top + 45
X3 = Pnode(Index + 1).Left + 45: Y3 = Pnode(Index + 1).Top + 45
End Select
'Erase in whiteness mode
If P = 1 Then ArtWork.AutoRedraw = True: ArtWork.DrawMode = 16
    ArtWork.DrawWidth = 2:
ArtWork.AutoRedraw = False
        ArtWork.Line (X1, Y1)-(X2, Y2): ArtWork.Line -(X3, Y3)
        ArtWork.Line (X2 - 45, Y2 - 45)-(X2 + 45, Y2 + 45), , B
        ArtWork.DrawMode = 6: ArtWork.DrawWidth = 1: p = 0
        ArtWork.Line (X1, Y1)-(X2, Y2): ArtWork.Line -(X3, Y3)
        ArtWork.Line (X2 - 45, Y2 - 45)-(X2 + 45, Y2 + 45), , B
End If
'Draw in inverse mode
ArtWork.Line (X1, Y1)-(X2, Y2): ArtWork.Line -(X3, Y3)
ArtWork.Line (X2 - 45, Y2 - 45)-(X2 + 45, Y2 + 45), , B
    X2 = X + Pnode(Index).Left
    Y2 = Y + Pnode(Index).Top
ArtWork.Line (X1, Y1)-(X2, Y2): ArtWork.Line -(X3, Y3)
ArtWork.Line (X2 - 45, Y2 - 45)-(X2 + 45, Y2 + 45), , B
If Index = 0 Then
  W(r - 1) = X + Pnode(Index).Left: N(r - 1) = Y + Pnode(Index).Top
  E(rr) = X + Pnode(Index).Left: S(rr) = Y + Pnode(Index).Top
Else
  W(r + 1) = X + Pnode(Index).Left: N(r + 1) = Y + Pnode(Index).Top
  E(r) = X + Pnode(Index).Left: S(r) = Y + Pnode(Index).Top
End If
End If
End Sub
```

If Button Then：如果鼠标按钮被按下，则继续。

LastIndex=(LastPolyPt(B)-FirstPolyPt(B))-1：用第一条线段编号减去最后一条线段编号，可得链中最后一个 Pnode 的正确索引值。

r=：正移动的 Pnode 左侧的 Pnode。

rr=：正移动的 Pnode 右侧的 Pnode。

If rr>LastIndex Then：对多边形中最后一个 Pnode 要特殊考虑。在本过程中，当移动一个 Pnode 时，需要有一条线段连接到 Pnode 控制对象的某一侧。最后一个 Pnode 相连的线段也连接着每一个 Pnode。可修改 rr 变量以指向第一个多边形点。

If Index=0 Then：多边形中第一个 Pnode 是一个特殊情况。第一个 Pnode 连接着与最后一个 Pnode 相连的最后一条线段。

Select Case Index：根据 Pnode 索引号，执行相应 Case 块。

Case 0：拖动 Pnode(0)。

X1=Pnode(LastIndex). Left+45：绘制一条连接最后一个 Pnode 的直线。

X3=Pnode(Index+1). Left+45：绘制连接着 Pnode(1)的一条直线。

Case LastIndex：拖动多边形中最后一个 Pnode。

X1=Pnode(Index-1). Left+45：绘制连接着前一个 Pnode 的一条直线。

X3=Pnode(0). Left+45：绘制连接 Pnode(0)的直线。

Case Else：拖动介于 Pnode(0)和最后一个 Pnode 之间的 Pnode。

X1=Pnode(Index-1). Left+45：连接并绘制 Pnode 左侧的直线。

X2=Pnode(Index+1). Left+45：连接并绘制 Pnode 右侧的直线。

下列 If/Then 语句只执行一次，用来擦除内存位图图象中的旧线段图象。

If P=1 Then：MouseMove 事件的第一次调用触发此 If/Then 代码块。

ArtWork. AutoRedraw：使用 AutoRedraw 属性可防止屏幕闪烁和图象跳动。将 AutoRedraw 打开即指向图片框控制对象的内存位图图象。将 DrawMode 置为 16(白色)，然后关掉 AutoRedraw，即可直接向图片框中绘图。

ArtWork. Line(X1,Y1)-(X2,Y2)：ArtWork. Line-(X3,Y3)：擦除两条线段。

ArtWork. Line(X2-45,Y2-45)-(X2+45,Y2+45),B：擦除所有 Pnode 鼠标漏点(鼠标移动一图象后遗留的一些象素点)。

ArtWork. DrawMode=6：将绘图模式置回 Inverse。

P=0：将标志清为 0，这样就不会重复此段代码。

ArtWork. Line：绘制两条线，再用 Line 语句的 B 选项绘制节点形状。

X2,Y2：改变末端点以取得移动效果。

ArtWork. Line：使用新的鼠标位置重绘直线。

If Index=0 Then：将要保存两条线段的新边界框位置。Pnode(0)需要特殊处理，因为其中一条线连接着最后一条多边形线段。

W(r-1)=X+Pnode(Index). Left：原直线的边界框只改变了一个角，所以需要修改第

一条线段的西、北值。

E(rr)=X+Pnode(Index). Left：第二条直线的边界框只改变了相对的一角，所以需要修改最后一条线段的东、南值。

Else：拖动除 Pnode(0)以外的 Pnode。

W(r+1),N(r+1),E(r),S(r)：除了使用不同的索引值来计算新的边界框坐标外，与上面描述的原则完全相同。

15.32 载入并重新定位多边形上的 Pnode

清单 15.27 中的 Poly_Nodes 过程为加入多边形的每条折线段载入新的 Pnode 图象控制对象。

清单 15.27 载入多边形上的新 Pnode

```
Sub Poly_Nodes (Index)
If Counter = 1 Then
    FillValue(B) = 1
    Node(0).Visible = False: Node(1).Visible = False
    Pnode(0).Left = Node(0).Left: Pnode(0).Top = Node(0).Top
    Pnode(0).Visible = True
    Load Pnode(1)
    Pnode(1).Left = Node(1).Left: Pnode(1).Top = Node(1).Top
    Pnode(1).Visible = True
End If
Total = Total + 1: B = Total
DrawOrder(B) = Total
FlagLine(B) = 0: PolyShape(B) = B: FirstPolyPt(B) = ZZ
W(B) = X1: E(B) = X2: N(B) = Y1: S(B) = Y2
LineWidth(B) = ArtWork.DrawWidth
r = NodeTotal 'add new node
If NodeTotal > 1 Then Load Pnode(r)
Pnode(r).Left = E(B) - 45: Pnode(r).Top = S(B) - 45
    NodeTotal = NodeTotal + 1
    Counter = Counter + 1
    Pnode(r).Visible = True
End Sub
```

If Counter=1 Then：counter 统计加入多边形的 Pnode 数目。当第一次向原直线对象添加新线段时，必须把这条直线转换为折线段。原直线的两端各有一个普通节点(Node(0),Node(1))，必须隐藏这些节点并用 Pnode 替代。

FillValue(B)=1：使对象为透明填充。

Node(0). Visible=False:Node(1). Visible=False：隐藏原来的 Node 图象控制对象。

Pnode(0). Left=Node(0). Left：以 Pnode(0)替代第一个 Node 图象控制对象。

Pnode(0). Visible=True：使隐藏的 Pnode 可见。

Load Pnode(1)：载入新 Pnode 以替代第二个 Node 图象控制对象。

Pnode(1). Left=Node(1). Left：放置第二个 Pnode。

Pnode(1). Visible=True：使新 Pnode 可见，因为所有新控制对象在载入时缺省为不可

见的。

Total＝Total＋1:B＝Total:对每条新折线段增加对象计数。

DrawOrder(B)＝Total:绘制顺序数组计数增 1。

FlagLine(B)＝0:保证没有折线段被标志为直线对象。

PolyShape(B)＝B:为新折线段指定编号。

FirstPolyPt(B)＝ZZ:把第一个多边形点的编号赋给每条新线段。

W(B)＝X1:E(B)＝X2:给予每条新线段一个边界框。

LineWidth(B)＝ArtWork. DrawWidth:指定线宽。

Y＝NodeTotal:记录载入的 Pnode 数目。

下列 If/Then 代码块给多边形添加额外的 Pnode 图象控制对象。

If NodeTotal＞1 Then Load Pnode(r):载入新 Pnode。

Pnode(r). Left＝E(B)－45:将 Pnode 置于正确位置。

NodeTotal＝NodeTotal＋1:Pnode 计数增 1。

Counter＝Counter＋1:Counter 计数增 1。

Pnode(r). Visible＝True:在定位新载入的 Pnode 图象控制对象后,使此 Pnode 可见。

15.33　重新载入多边形上的 Pnode

清单 15.28 中的 Poly_Reload 过程将围着选中的多边对象重新定位 Pnode。必须考虑到多边形可能是折线和曲线段的混合。放置 Pnode 时对线段使用存放于边界框(W,N,E,S)数组中的值,而对曲线段则用存放于曲线控制点(P1,P2)数组中的值。

清单 15.28　当选中多边形时重新载入 Pnode

```
Sub Poly_Reload ()
    If StatusLabel.Caption = "No color Found" Then Exit Sub
    If StatusLabel.Caption = "No Object Found" Then Exit Sub
    I = FirstPolyPt(B): rr = 1
On Error GoTo AlreadyLoaded
If FirstPolyPt(B) = FlagBézier(i) Then
    Pnode(0).Left = p1X(i) - 45: Pnode(0).Top = p1Y(i) - 45
Else
    Pnode(0).Left = W(i) - 45: Pnode(0).Top = N(i) - 45
End If
For r = FirstPolyPt(i) + 1 To LastPolyPt(i) - 1'add new nodes
    Load Pnode(rr)
If FlagBézier(r) = r Then
    Pnode(rr).Left = p1X(r) - 45: Pnode(rr).Top = p1Y(r) - 45
Else
    Pnode(rr).Left = W(r) - 45: Pnode(rr).Top = N(r) - 45
End If
Pnode(rr).Visible = True: rr = rr + 1
Next r
NodeTotal = rr - 1: Pnode(0).Visible = True
Toggle = True
AlreadyLoaded:
    FCP = FirstPolyPt(B)
    LCP = LastPolyPt(B) - 1
    If FlagPolyLine(FCP) = FCP Then
    Load Pnode(rr)
If FlagBézier(LCP) = LastPolyPt(LCP) - 1 Then
    Pnode(rr).Left = p2X(LCP) - 45: Pnode(rr).Top = p2Y(LCP) - 45
Else
    Pnode(rr).Left = E(LCP) - 45: Pnode(rr).Top = S(LCP) - 45
End If
    Pnode(rr).Visible = True
End If
Exit Sub
End Sub
```

If StatusLabel. Caption：在 Find_object 过程中(清单 15.11)，StatusLabel. Caption 将指示是否找到一个对象。如果没有找到任何对象则退出此过程。

I=FirstPolyPt(B);rr=1：变量 I 相当于多边形的第一条线段。变量 rr 等于要载入的第一个 Pnode 索引。

On Error Goto AlreadyLoaded：如果试图载入一个已载入并显示于桌面上的新控制对象将会导致错误。为了防止程序停止，在发生错误时将跳转到 AlreadyLoaded 语句行。

If FirstPolyPt(B)=FlagBézier(I) Then：第一条多边形线段是 Bézier 曲线，故使用曲线数组。

Pnode(0). Left=p1X(I)-45：用曲线的控制点值定位第一个 Pnode。

Else：第一条多边形线段是直线段，故使用边界框数组。

Pnode(0). Left=W(I)-45：用线段的拐角位置定位 Pnode。

下面将把剩下的 Pnode 图象控制对象载入多边形。

For r=FirstPolyPt(I)+1 To LastPolyPt(I)-1：从第二个多边形线段开始直到最后一个线段点加入 Pnode。

Load Pnode(rr)：从 Pnode(1)开始载入新 Pnode。

If FlagBézier(r)=r Then：如果线段是曲线，则使用曲线数组值。

Pnode(rr). Left=p1X(r)-45：在曲线的控制点处定位 Pnode。

Else：多边形线段是直线，故使用边界框数组值。

Pnode(rr). Left=W(r)-45：使用线段的拐角位置定位 Pnode。

Pnode(rr). Visible=True;rr=rr+1：所有新控制对象默认为不可见，故使之可见。

Next r：重复 For/Next 代码块直到所有 Pnode 均已载入并正确定位。

NodeTotal=rr-1：保存刚载入的 Pnode 数目。

Toggle=True：本标志通知 Pnode Mouse_Move 事件(清单 15.18)ArtAPI 程序已准备好移动任何多边形线段。

AlreadyLoaded：如果发生错误则跳转至此处。

下列代码用于开多边形。因为开多边形不具有 Polygon 形状中的闭合线段，所以不需要放置链中最后一个 Pnode。必须为开多边形的最后一条线段手工赋以 Pnode。

FCP＝FirstPolyPt(B)：等于第一条线段的起始点。

LCP＝LastPolyPt(B)：等于最后一条线段的起始点。

If FlagPolyLine(FCP)：本标志通知程序对象是否是开多边形。

Load Pnode(rr)：如果确为开多边形则载入一个 Pnode。

If FlagBézier(LCP)：如果开多边形的最后一条线段为曲线，执行后续代码。

Pnode(rr). Left＝p2X(LCP)－45：在曲线的终止控制点处定位 Pnode。

Else：最后一条线段是直线，故执行后续代码。

Pnode(rr). Left＝E(LCP)－45：在直线终点放置 Pnode。

Pnode(rr). Visible＝True：使新 Pnode 控制对象可见。

15.34　在编辑多边形线段之后

在编辑多边形时，Pnode 在鼠标移动事件期间被隐藏。在清单 15.29 中，可调用 Poly_Update 将 Pnode 置于新位置。

清单 15.29　将被隐藏的 Pnode 置于新位置

```
Sub Poly_Update (Index%, Button%, Shift%, X As Single, Y As Single)
    XXX = (X + Pnode(Index).Left) - 45
    YYY = (Y + Pnode(Index).Top) - 45
    Pnode(Index).Left = XXX
    Pnode(Index).Top = YYY
    Pnode(Index).Visible = True
StatusLabel.Caption = "Node " + Format $ (Index) + " (" +
            Format $ (Pnode(Index).Left \ PixelX) + " - " +
                Format $ (Pnode(Index).Top \ PixelY) + " )"
    UpdateObjects
End Sub
```

XXX＝(X＋Pnode(Index). Left)－45：重定位 Pnode 到新 X 坐标。

YYY＝(Y＋Pnode(Index). Top)－45：重定位 Pnode 到新的 Y 坐标。

UpdateObjects：重绘桌面上所有对象。

15.35　调整多边形大小和拉伸多边形

清单 15.30 中的 Poly_Stretch 过程重新定位多边形中所有折线和曲线段的 X，Y 坐标。本例程同第 3 章“Bézier 曲线”中的 Bézier 拉伸过程完全相同。不过必须加入 For/Next 语句以遍历所有多边形线段。

Poly_Stretch 过程见清单 15.30。一旦熟悉了它的工作原理，就可以很容易地把各部分合并起来以缩短代码。

清单 15.30　调整多边形大小或翻转多边形

```
Sub Poly_Stretch (Index%, Button%, Shift%, X As Single, Y As Single)
Select Case Index
    Case 0, 1, 2
        percentage! = (Y - YY1) / (YY2 - YY1): ANCHOR = YY2
    Case 3
        percentage! = (X - XX2) / (XX1 - XX2): ANCHOR = XX1
        Case 4, 5, 6
            percentage! = (Y - YY2) / (YY1 - YY2): ANCHOR = YY1
        Case 7
            percentage! = (X - XX1) / (XX2 - XX1): ANCHOR = XX2
    End Select
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''
    For i = FirstPolyPt(B) To LastPolyPt(B) - 1
    Select Case Index
        Case 0, 1, 2, 4, 5, 6
            Bouy1 = N(i): Bouy2 = S(i)
            p1XY = p1Y(i): p2XY = p2Y(i)
            h1XY = h1Y(i): h2XY = h2Y(i)
        Case 3, 7
            Bouy1 = W(i): Bouy2 = E(i)
            p1XY = p1X(i): p2XY = p2X(i)
            h1XY = h1X(i): h2XY = h2X(i)
    End Select
        Bouy1 = ((ANCHOR - Bouy1) * percentage!) + Bouy1
        Bouy2 = ((ANCHOR - Bouy2) * percentage!) + Bouy2
        Boat1 = p1XY: Boat2 = p2XY
        Boat3 = h1XY: Boat4 = h2XY
        Boat1 = ((ANCHOR - Boat1) * percentage!) + Boat1
        Boat2 = ((ANCHOR - Boat2) * percentage!) + Boat2
        Boat3 = ((ANCHOR - Boat3) * percentage!) + Boat3
        Boat4 = ((ANCHOR - Boat4) * percentage!) + Boat4

    Select Case Index
        Case 0, 1, 2, 4, 5, 6
            N(i) = Bouy1: S(i) = Bouy2
            p1Y(i) = Boat1: p2Y(i) = Boat2
            h1Y(i) = Boat3: h2Y(i) = Boat4
        Case 3, 7
            W(i) = Bouy1: E(i) = Bouy2
            p1X(i) = Boat1: p2X(i) = Boat2
            h1X(i) = Boat3: h2X(i) = Boat4
    End Select
    Next i
    '''''corners
    Select Case Index
            Case 0, 6 'Left corners
                percentage! = (X - XX1) / (XX2 - XX1): ANCHOR = XX2
            Case 2, 4 'Right corners
                percentage! = (X - XX2) / (XX1 - XX2): ANCHOR = XX1
            Case Else
                Exit Sub
    End Select
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''
    For i = FirstPolyPt(B) To LastPolyPt(B) - 1
        Bouy1 = W(i): Bouy2 = E(i)
        Bouy1 = ((ANCHOR - Bouy1) * percentage!) + Bouy1
        Bouy2 = ((ANCHOR - Bouy2) * percentage!) + Bouy2
        Boat1 = p1X(i): Boat2 = p2X(i)
        Boat3 = h1X(i): Boat4 = h2X(i)
        Boat1 = ((ANCHOR - Boat1) * percentage!) + Boat1
        Boat2 = ((ANCHOR - Boat2) * percentage!) + Boat2
        Boat3 = ((ANCHOR - Boat3) * percentage!) + Boat3
        Boat4 = ((ANCHOR - Boat4) * percentage!) + Boat4

        W(i) = Bouy1: E(i) = Bouy2
        p1X(i) = Boat1: p2X(i) = Boat2
        h1X(i) = Boat3: h2X(i) = Boat4
    Next i
    End Sub
```

请参见第 3 章“Bézier 曲线”来了解 Poly_Stretch 例程的内部工作机制。

15.36　使用测试按钮

清单 15.31 中的 TestButton_Click 过程将重绘当前选中的多边形。本例程使用 Visual Basic Line 语句来绘制多边形并可替代 API polyline 语句。不过，不能用颜色填充多边形，因为 Visual Basic 语言不支持 Paint 语句。这种情况下仍需使用 API 函数 Floodfill 来填充屏幕上的区域。

清单 15.31　绘制多边形的路径轮廓

```
Sub TestButton_Click ()
ArtWork.DrawWidth = 8: ArtWork.DrawMode = 13
ArtWork.DrawStyle = 0: rr = 1
For r = FirstPolyPt(B) To LastPolyPt(B) - 1
    X1 = W(r): Y1 = N(r): X2 = E(r): Y2 = S(r)
''''''''''''''''''''''''''''''''''''

If FlagBézier(r) = r Then
    p1.X = p1X(r): p1.Y = p1Y(r): p2.X = p2X(r): p2.Y = p2Y(r)
    h1.X = h1X(r): h1.Y = h1Y(r): h2.X = h2X(r): h2.Y = h2Y(r)
    ArtWork.ForeColor = QBColor(Rnd * 14)
    BezDraw
    GoTo SkipLine:
End If
    ArtWork.Line (X1, Y1)-(X2, Y2), QBColor(Rnd * 14)
SkipLine:
Next r
    ArtWork.DrawWidth = 1: ArtWork.DrawMode = 6
End Sub
```

For r=FirstPolyPt(B) To LastPolyPt(B)−1：从第一条多边形线段到最后一条多边形线段，执行后续语句：

X1=W(r)；Y1=N(r)；X2=E(r)；Y2=S(r)：取得每条线段的始、末点。

If FlagBézier(r)=r Then：如果线段为曲线，执行后续代码：

p1.X=p1X(r)：取得曲线的 4 个控制点。

BezDraw：调用此过程绘制 **Bézier** 典线。

GoTo SkipLine：已画完曲线，故跳转至 SkipLine，可以不执行后续的 Line 语句。

ArtWork. Line：如果是折线则绘制之。

SkipLine：如果多边形线段是曲线则跳转此处。

Next r：对下一条多边形线段重复所有工作。

15.37 将对象移到前面或后面

清单 15.32 中的 To_Front_Back 过程用于把某对象移到所有其他对象的前面或后面。第 4 章的 Artisan 程序通过改变对象的次序及相关属性来做到这一点。现在只需一个短小的循环就能做到这一点。无需切换每个对象的编号及属性，只要简单地修改其初始绘制次序即可。这样做只需添加一个数组(DrawOrder)来记录绘图次序。清单 15.35 中的 UpdateObjects 过程根据存放于 DrawOrder 数组中的次序使用了一个 For/Next 语句来绘制每个对象。

清单 15.32 将某对象移至所有其他对象的前面或后面

```
Sub To_Front_Back (Index As Integer)
If B = 0 Then Beep: Exit Sub
If PolyShape(B) = B Then B = FirstPolyPt(B)
Select Case Index
    Case 0 ' To Front
        For Objectnumber = 1 To Total
            Z = DrawOrder(Objectnumber)
            If Z = B Then Exit For
        Next Objectnumber     'Objectnumber will highlighted object
Temp = DrawOrder(Objectnumber)
    For Repeat2 = Objectnumber To Total
        NextA = Repeat2 + 1
        DrawOrder(Repeat2) = DrawOrder(NextA)
    Next Repeat2
DrawOrder(Total) = temp
    Case 1 ' To back
        For Objectnumber = 1 To Total
            Z = DrawOrder(Objectnumber)
            If Z = B Then Exit For
        Next Objectnumber     'Objectnumber will highlighted object
Temp = DrawOrder(Objectnumber)
    For Repeat2 = Objectnumber To 2 Step -1
        NextA = Repeat2 - 1
        DrawOrder(Repeat2) = DrawOrder(NextA)
        Next Repeat2
  DrawOrder(1) = temp
End Select
        UpdateObjects
End Sub
```

If B=0 Then Beep：如果没有选择对象，则退出此过程。

If PolyShape(B)=B Then：如果对象是多边形，则必须得到第一个多边线段以指向整个对象。

Select Case Index；根据所选菜单项，执行后续代码：

Case 0：选择了把对象移到所有其他对象的前面。

For Objectnumber=1 To Total：对每个对象做以下工作：

Z=DrawOrder(Objectnumber)：变量 Z 代表赋给 DrawOrder 数组中每个下标的对象编号。每个值都是赋给对象的编号。如果不使用此过程，则对象编号是有序的(即数组下标等于对象编号)。每次向前或向后移动对象都会打乱数组。

If Z＝B Then Exit For：B 等于当前选中对象的编号。如果数组已打乱以至于不知道这个值在数组中的位置，则可遍历数组的下标。如果找到对象编号即达到了目的。

Next Objectnumber：遍历索引以找到当前对象的编号。

Temp＝DrawOrder(Objectnumber)：已找到对象的编号，故保存放有对象编号的下标值。

For Repeat2＝Objectnumber To Total：从刚找到的索引号开始，直到 DrawOrder 数组的最后一个索引，做以下工作：

NextA＝Repeat2＋1：保存下一个更高索引号。

DrawOrder(Repeat2)＝DrawOrder(NextA)：把下一个索引中的对象编号放入当前索引。

Next Repeat2：继续获取下一索引中的对象号。

DrawOrder(Total)＝Temp：当到达索引顶端时，更高的索引中不包含任何东西。幸好已保存了当前对象编号，这样对象将显示于最上层。如果你擅长堆牌的话，可以把对象向前或向后移动任意步。

Case 1：选择了将对象移到所有其他对象的后面。

这部分同前面 Case 0 代码块完全一样。唯一不同的是向后循环。

For Repeat2＝Objectnumber To 2 Step －1：仅循环至第 2 个索引位置以把当前对象放入第 1 个索引中。Step －1 关键字使得向后计数。

NextA＝Repeat2－1：从较低的索引中取出对象编号。

UpdateObjects：重绘新打乱的所有对象。

15.38　使用轮廓图象选择一对象

清单 15.33 中的 UpdateArtMask 过程类似于 UpdateObject 过程（清单 15.35），不同的是本过程使用 4 个象素的绘图宽度向 ArtMask 图象控制对象中绘制单个对象。Find_Object 过程（清单 15.11）将调整控制对象大小以节省空间，然后寻找对象的轮廓，如果找到则选择此对象。可以将 ArtMask 的 AutoRedraw 属性设为 True，可见属性设为 False 来模仿屏外绘图从而做到这一点。

第 3 章 Artisan 程序中对象的寻找是用了一个气泡型数组，搜索所有边界框直到找到一个位于鼠标之上的边界框。当几个边界框（或对象）互相交叠时这个办法就行不通，而且这样就不可能拾取或移动被可选择对象覆盖的对象。

清单 15.33　向内存位图绘制对象

```
Sub UpdateArtMask ()
    ArtMask.PSet (0, 0)
        If PolyShape(B) = B Then
            If FlagBézier(B) = 0 Then FlagLine(B) = B
        End If
ArtMask.DrawWidth = 4
FlagArtMask = True
X1 = W(B) - ArtMask.Left: Y1 = N(B) - ArtMask.Top
```

```
X2 = E(B) - ArtMask.Left: Y2 = S(B) - ArtMask.Top
Select Case B
    Case FlagCircle(B)      'Ellipse
        r = Ellipse(ArtMask.hDC, X1 \ PixelX, Y1 \ PixelY, X2
                    \ PixelX, Y2 \ PixelY)
    Case FlagLine(B)            'Single polyline
        Npt(1).X = X1 \ PixelX: Npt(1).Y = Y1 \ PixelY
        Npt(2).X = X2 \ PixelX: Npt(2).Y = Y2 \ PixelY: PolyPts = 2
        r = Polyline(ArtMask.hDC, Npt(1), PolyPts)
    Case FlagBox(B)'rectangle
        API = Rectangle(ArtMask.hDC, X1 \ PixelX, Y1
                \ PixelY, X2 \ PixelX, Y2 \ PixelY)
    Case FlagBézier(B)      'Bézier curve
        I = 1 : BB = B
p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
        Poly_curves I
        API = Polyline(ArtMask.hDC, Npt(1), 33)
        SkipaCurve:
    Case FlagText(B) 'text
        ArtMask.CurrentX = OFFSET(B) - 15: ArtMask.CurrentY = N(B) - 15
        SaveScaleLeft = ArtMask.ScaleLeft
        ArtMask.ScaleLeft = ArtMask.Left - ArtMask.CurrentX
        ArtMask.FontSize = TextSize(B)
        ArtMask.FontName = TextFont(B)
        ArtMask.FontSize = TextSize(B) 'SMALL FONT CALL
        ArtMask.ForeColor = 0
        ArtMask.FontBold = TextBold(B)
        ArtMask.FontItalic = TextItalic(B)
        wFlag = FlagAlign(B)
        TA = SetTextAlign(ArtMask.hDC, wFlag)
        ArtMask.Print StoredText(B)
        ArtMask.ScaleLeft = SaveScaleLeft
End Select
'''''
        FlagArtMask = False
    If PolyShape(B) = B Then FlagLine(B) = 0
End Sub
```

ArtMask.PSet(0,0)：有时会发现 API 图形函数不能马上在 Visual Basic 控制对象中绘图，要校正这一点就需要取得要绘图的控制对象设备环境图柄。在使用 Basic 语言图形、字体或打印语句时，Visual Basic 自动完成这些工作。通过向控制对象中放置一个象素点，即可拥有此控制对象的设备环境引用。使用 0,0 坐标是因为在此位置将不会看到此象素。

If PolyShape(B)=B Then：如果正在绘制多边形的一部分，则执行后续工作：

If FlagBézier(B)=0 Then FlagLine(B)=B：UpdateObject 过程绘制整个多边形要使用 FlagLine 例程之一。但此处只需绘制单独的折线段。与其使用很长的例程来拾取并选择正确的线段。不如把所有折线重置为 FlagLine 对象。在一条折线段绘制完后可以将 FlagLine 置 0 来删去此 FlagLine 对象。

ArtMask.DrawWidth=4：鼠标只寻找对象的轮廓，故使被遮蔽的对象为 4 个象素宽以

便鼠标能更容易地找到它。

FlagArtMask＝True：本标志通知 Poly_Curves 过程（清单 15.22）要在 ArtMask 图片框控制对象中绘制曲线。

X1＝W(B)－ArtMask.Left：Y1，X2，Y2：所有这些值都用来将图形置于 ArtMask 控制对象中正确的位置并精确匹配显示于桌面上的图形位置。

Select Case B：请参考 UpdateObjects 过程（清单 15.37）来了解本过程中 Select/Case 代码块中出现的 API 图形语句。本代码块后面的语句是本过程中最后一条语句，用来重置前面修改过的对象标志。

If PolyShape(B)＝B Then FlagLine(B)＝0：删去直线对象标志。

15.39　在桌面上绘制所有对象

清单 15.34 中的 UpdateObject 过程同第 4 章 Artisan 程序中的 UpdateObject 过程基本相同。主要的区别是在 ArtAPI 程序中，使用 API 绘图函数来向桌面上绘制对象。

清单 15.34　在桌面上绘制所有对象

```
Sub UpdateObjects ()
StoreB = B: ArtWork.DrawStyle = Solid
ArtWork.DrawMode = Copy_Pen: ArtWork.AutoRedraw = True
If Draw = True Then GoTo SkipErase
    ArtWork.DrawWidth = 20
    ArtWork.Line (W(B), N(B))-(E(B), S(B)), White, BF
    ArtWork.Line (XX1, YY1)-(XX2, YY2), White, BF
If B > 0 And PolyShape(B) = B Then
    ArtWork.Line (W(0), N(0))-(E(0), S(0)), White, BF
    Poly_Handles
    ArtWork.Line (W(0), N(0))-(E(0), S(0)), White, BF
End If
UpdateDesktop
For Repeat = 1 To Total
    B = DrawOrder(Repeat)
    If B = 0 Then B = Repeat
If LineValue(B) > 0 Then ArtWork.DrawStyle = 5 Else
                ArtWork.DrawStyle = Solid
SkipErase:
X1 = W(B): Y1 = N(B): X2 = E(B): Y2 = S(B)
ArtWork.FillColor = PaintColor(B): ArtWork.FillStyle = FillValue(B):
    ArtWork.DrawWidth = LineWidth(B)
rr& = ArtWork.ForeColor: ArtWork.ForeColor = OutLineColor(B)
```

注意：　*在下列说明之后会继续列出后续代码清单。*

StoreB＝B：现在将使用 For/Next 语句来遍历所有对象。在循环中 B 的值将改变，所以要把当前对象值临时存储在变量 StoreB 中。

ArtWork.DrawStyle：以默认的 Solid（实心）模式开始绘图。

ArtWork.DrawMode：以默认的 Copy_Pen 绘制模式重绘对象。

If Draw：无论何时绘制或拉伸了一个位于其他所有层上的对象，此变量将置为 True，

这样就不需要重绘其他对象。即可以跳过 For/Next 语句直接转到 SkipErase 来只绘出最上层的对象。

下面 3 行用来清除对象边界框的上一次位置。

ArtWork.DrawWidth：将边界框边框宽度设为 20 个象素以使此对象具有宽轮廓，否则无法清除下面的区域。

ArtWork.Line(W(B),N(B))－(E(B),S(B))：此为当前边界框尺寸，故填充白色即可清除此区域。

ArtWork.Line(XX1,YY1)－(XX2,YY2)：对象移动或拉伸前旧的边界框位置。也可通过填充白色来清除此区域。

下面 3 行用来清除多边形边界框的上一次位置。

If B>0 And PolyShape(B)＝B Then：如果当前对象是多边形，则执行后续代码。

ArtWork.Line (W(0),N(0))－(E(0),S(0))：此为要擦除的多边形的当前边界框尺寸。对任何多边形边界框 W,N,E,S 数组的 0 下标值互相颠倒。

Poly_Handles：在拉伸后计算新的边界框区域。

ArtWork.Line(W(0),N(0))－(E(0),S(0))：擦去位于新多边形图象下面的区域。

UpdateDeskTop：在程序开头置于桌面中央的页面实际由两个矩形组成，一个填充黑色，另一个填充白色并略向左移。可调用过程 UpdateDeskTop 来重绘此页面。

下面 3 行用来按照 DrawOrder 数组指定的次序绘制所有对象。

For Repeat＝1 To Total：使用 For/Next 语句替换了第 4 章 Artisan 程序中的 Do/While 循环语句，这样可以遍历所有绘图例程。

B＝DrawOrder(Repeat)：变量 B 即为将要绘制的对象。DrawOrder 数组的下标从 0 直到对象的总数。每个下标均包含对象的编号，此编号是否有序取决于是否使用过 To_Front_Back 过程。

If B＝0 Then B＝Repeat：为防止 DrawOrder 数组发生混乱情况的保护手段。

If LineValue(B)：使对象的 LineValue 等于 0，即可隐藏此对象的轮廓，而不能把 DrawWidth 设为 0。如果想要一个不带轮廓的对象，可将对象的 LineValue 变量置 0，即可使 ArtWork 图片框的 DrawStyle 在绘制轮廓时变为透明(1)。

SkipErase：如果标志 Draw 是 True 则跳过 For/Next 循环。

X1＝W(B)：Y1＝N(B)：指定将被圆或方框对象使用的坐标。

ArtWork.FillColor,etc.：为将要绘制的对象指定合适的刷子和绘图宽度。

rr&＝ArtWork.ForeColor：保存桌面当前画笔(ForeColor)以便绘制对象后置回初始颜色。

清单15.35 在桌面上绘制对象(续)

```
  Select Case B
      Case FlagCircle(B)          'Ellipse
          r = Ellipse(ArtWork.hDC, X1 \ PixelX, Y1 \ PixelY, X2
                            \ PixelX, Y2 \ PixelY)
      Case FlagLine(B)            'Single polyline
          Npt(1).X = X1 \ PixelX: Npt(1).Y = Y1 \ PixelY
          Npt(2).X = X2 \ PixelX: Npt(2).Y = Y2 \ PixelY: PolyPts = 2
          r = Polyline(ArtWork.hDC, Npt(1), PolyPts)
      Case FlagPolyLine(B) 'Open multi-Polyline Shape
          fp = FirstPolyPt(B): LP = LastPolyPt(B) - 1: I = 1
  For BB = fp To LP
      If BB = FlagBézier(BB) And BB = PolyShape(BB) Then
          p1.X = p1X(BB): p1.Y = p1Y(BB): p2.X = p2X(BB): p2.Y = p2Y(BB)
          h1.X = h1X(BB): h1.Y = h1Y(BB): h2.X = h2X(BB): h2.Y = h2Y(BB)
          Poly_curves I
      Else
          Npt(I).X = W(BB) \ PixelX: Npt(I).Y = N(BB) \ PixelY: I = I + 1
      End If
  Next BB
  Npt(I).X = E(BB - 1) \ PixelX: Npt(I).Y = S(BB - 1) \ PixelY
  If FlagBezier(BB - 1) = BB - 1 Then Npt(I).X = p2.X \ PixelX:
                                Npt(I).Y = p2.Y \ PixelY
          PolyPts = I
          API = Polyline(ArtWork.hDC, Npt(1), PolyPts)
  ''''''''''''''''''''''''''''''''''''''''''''
      Case FlagPoly(B) 'Closed polygon shape
          fp = FirstPolyPt(B): LP = LastPolyPt(B) - 1: I = 1
  For BB = fp To LP
      If BB = FlagBézier(BB) And BB = PolyShape(BB) Then
          p1.X = p1X(BB): p1.Y = p1Y(BB): p2.X = p2X(BB): p2.Y = p2Y(BB)
          h1.X = h1X(BB): h1.Y = h1Y(BB): h2.X = h2X(BB): h2.Y = h2Y(BB)
          Poly_curves I
      Else
          Npt(I).X = W(BB) \ PixelX: Npt(I).Y = N(BB) \ PixelY: I = I + 1
      End If
  Next BB
          API = SetPolyFillMode(ArtWork.hDC, 1): PolyPts = I - 1
          API = Polygon(ArtWork.hDC, Npt(1), PolyPts)
      Case FlagBox(B)             'rectangle
          API = Rectangle(ArtWork.hDC, X1 \ PixelX, Y1 \ PixelY, X2
                            \ PixelX, Y2 \ PixelY)
      Case FlagBézier(B)          'Bézier curve
          If FlagBézier(B) = PolyShape(B) Then GoTo SkipCurve
  p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
  h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
          ArtWork.ForeColor = OutLineColor(B)
          BezDraw
          SkipCurve:
      Case FlagText(B)        'text
          ArtWork.CurrentX = OFFSET(B): ArtWork.CurrentY = N(B)
          SaveScaleLeft = ArtWork.ScaleLeft
```

```
            ArtWork.ScaleLeft = -ArtWork.CurrentX

            ArtWork.FontSize = TextSize(B)
            ArtWork.FontName = TextFont(B)
            ArtWork.FontSize = TextSize(B) 'SMALL FONT CALL
            ArtWork.ForeColor = PaintColor(B)
            ArtWork.FontBold = TextBold(B)
            ArtWork.FontItalic = TextItalic(B)
            wFlag = FlagAlign(B)
            TA = SetTextAlign(ArtWork.hDC, wFlag)
            ArtWork.Print StoredText(B)
            ArtWork.ScaleLeft = SaveScaleLeft
    End Select
```

注意：*在下列说明之后会继续列出代码清单。*

Select Case B：变量 B 代码对象的当前边界框。

Case FlagCircle：如果当前边界框包括一个圆，则：

X1、Y1、X2、Y2：传给 API 函数 Ellipse 的边界框位置。

Case FlagLine(B)：对象为直线。使用固定的 PolyPts 值 2 是因为只需传给 API 函数 Polyline 两个 X，Y 坐标。Npt 数组中包含传给函数的 X，Y 坐标值。

Case FlagPolyLine(B)：绘制开折线形对象。

For BB=fp To LP：从第一条折线段到最后一条折线段的起点，执行以下代码：

If BB=FlagBézier(BB) And BB=PolyShape(BB) Then：线段为 Bézier 曲线，故获取曲线的 4 个控制点。

p1.X=p1X(BB),etc.：取得 4 个曲线控制点。

Poly_curves I：计算并绘制 Bézier 曲线。参数 I 将记录赋给 API 函数 Polyline 所用的 Npt 数组的索引数组数目。

Else：线段为直线，故获取始、末点。

Npt(I).X=W(BB)\PixelX：为 Npt 数组指定正确的 X 和 Y 坐标。被保存的值是 twip 单位，所以要转换成象素单位。最后，将 Npt 索引计数(I)加 1。

Next BB：对多边形中所有线段重复上述过程。

Npt(I).X=E(BB-1)\PixelX：一个开多边形没有闭多边形自动闭合末端点的好处。必须为开折线形手工设置最后一点的 X 和 Y 坐标。

If FlagBézier(BB-1)=BB-1 Then：如果最后一点连着一条 Bézier 曲线，则执行 If/Then 代码块。

PolyPts=I：I 变量等于要绘制的索引 X，Y 点的总数。

API=Polyline(ArtWork.hDC,Npt(1),PolyPts)：绘制开折线形。

Case FiagPoly(B)：绘制所有闭合多边形。

除了 PolyPts 计数之外，这段例程很像 FlagPolyLine 的情况。最后一条线段自动连接到头一条线段以形成闭合多边形。

API=SetPolyFillMode(ArtWork.hDC,1)：可以选择两种多边形填充方式之一，即“间隔方式”或“卷绕方式”。请参见第 13 章“节点连接程序”(清单 13.9)。

PolyPts=I−1：减去最后一个索引点，因为这个点对形成多边形没有用处。

API=Polygon(ArtWork. hDC,Npt(1),PolyPts)：绘制闭合多边形。

Case FlagBox(B)：绘制矩形框对象。

X1,Y1,X2,Y2：需要传给 API 函数 Rectangle 的边界框坐标。

Case FlagBézier：对象是 Bézier 曲线，所以要获取 4 个控制点，然后把 ArtWork 的 ForeColor 改为曲线的轮廓颜色。BezDraw 过程使用这个颜色属性来绘制 Bézier 曲线。本例程只绘制单条 Bézier 曲线并且可以和 FlagPolyLine 例程合并起来。

Case FlagText(B)：本程序没有使用任何 API 文本函数。所有文本都使用 Basic 语言的语句来人工格式化。细节见第 5 章的 UpdateObjects 过程。可以参考第 12 章 APItext 程序中的标准 API 文本格式化过程。Artwork 的文本尺寸被置了两次，一次在字体名之前，一次在此之后。Windows 在显示字体小于 8 点阵时使用位图字体。取消字体尺寸(或字体名)将把控制的字体属性置回初始值。

清单 15.36　在桌面上绘制对象(续)

```
ArtWork. ForeColor=rr&:ArtWork. FillColor=0:ArtWork. FillStyle=1
If Draw=True Then Draw=False:ArtWork. AutoRedraw=False:
                              B=StoreB: Exit Sub
   B=B+1
Next Repeat
   B=StoreB
   ArtWork. AutoRedraw=False
   ArtWork. DrawWidth=1:ArtWork. DrawStyle=Solid
If DeleteObject=True Then Exit Sub
If CurrentTool=1 Or CurrentTool=2 Or CurrentTool=10 Then Exit Sub
If SideRuler=True Or TopRuler=True Then Exit Sub
      Handles
      Nodes
End Sub
```

请参考第 5 章中 Artisan 程序的 UpdateObjects 过程(清单 5.16)，以详细了解清单 15.36 中的剩余代码。

15.40　打印所有对象

除了使用人工定标值以向打印机页打印图形外，清单 15.37 中的 WinPrint 过程和清单 15.34 中的 UpdateObjects 过程完全相同。

清单 15.37　向打印机页面打印所有对象

```
Sub WinPrint ()
    On Error GoTo PrintErr:
CMDialog1. Flags = &H2& Or &H4& Or &H8& Or &H100000
CMDialog1. Action = 5
NumCopies = CMDialog1. Copies
Screen. MousePointer = 11
For r = 1 To NumCopies
```

```
        Dim RightM As Single
        Dim BottomM As Single
LeftM = PageX1: TopM = PageY1 'Adjust Margins
APIH = GetDeviceCaps(Printer.hDC, 8)
APIV = GetDeviceCaps(Printer.hDC, 10)
HPrinterDPI! = APIH / (PageX2 - PageX1)
VPrinterDPI! = APIV / (PageY2 - PageY1)
PageScale! = Printer.ScaleWidth / (PageX2 - PageX1)
RightM = HPrinterDPI!        'Adjust Margins
BottomM = VPrinterDPI!
    FlagPrinter = True
    Printer.PSet (0, 0)
'''''''''''''''
For Repeat = 1 To Total
    B = DrawOrder(Repeat)
    If B = 0 Then B = Repeat
If LineValue(B) > 0 Then Printer.DrawStyle = 5 Else
                      Printer.DrawStyle = Solid
X1 = (W(B) - LeftM) * RightM: Y1 = (N(B) - TopM) * BottomM:
X2 = (E(B) - LeftM) * RightM: Y2 = (S(B) - TopM) * BottomM
Printer.FillColor = PaintColor(B): Printer.FillStyle = FillValue(B)
If LineWidth(B) > 1 Then Printer.DrawWidth = LineWidth(B) * 5 Else
                      Printer.DrawWidth = LineWidth(B)
    rr& = Printer.ForeColor: Printer.ForeColor = OutLineColor(B)
Select Case B
    Case FlagCircle(B)      'Ellipse
r = Ellipse(Printer.hDC, X1, Y1, X2, Y2)
    Case FlagLine(B)            'Single polyline
Npt(I).X = X1: Npt(I).Y = Y1: FLP = I: I = I + 1
Npt(I).X = X2: Npt(I).Y = Y2: I = I + 1
    r = Polyline(Printer.hDC, Npt(FLP), 2)
''''''''''''''''''''''''''''''''''''''''''
    Case FlagPolyLine(B) 'Open multi-Polyline Shape
fp = FirstPolyPt(B): LP = LastPolyPt(B) - 1: I = 1
For BB = fp To LP
    If BB = FlagBézier(BB) And BB = PolyShape(BB) Then
        p1.X = p1X(BB): p1.Y = p1Y(BB): p2.X = p2X(BB): p2.Y = p2Y(BB)
        h1.X = h1X(BB): h1.Y = h1Y(BB): h2.X = h2X(BB): h2.Y = h2Y(BB)
        Poly_curves I
    Else
        Npt(I).X = (W(BB) - LeftM) * RightM
        Npt(I).Y = (N(BB) - TopM) * BottomM: I = I + 1
    End If
Next BB
        Npt(I).X = (E(BB - 1) - LeftM) * RightM
        Npt(I).Y = (S(BB - 1) - TopM) * BottomM
        If FlagBézier(BB - 1) = BB - 1 Then Npt(I).X = (p2.X - LeftM) * RightM:
                      Npt(I).Y = (p2.Y - TopM) * BottomM
        PolyPts = i
            API = Polyline(Printer.hDC, Npt(1), PolyPts)
''''''''''''''''''''''''''''''''''''''''
    Case FlagPoly(B) 'Closed polygon shape
fp = FirstPolyPt(B): LP = LastPolyPt(B) - 1: I = 1
  For BB = fp To LP
```

```
    If BB = FlagBézier(BB) And BB = PolyShape(BB) Then
        p1.X = p1X(BB): p1.Y = p1Y(BB): p2.X = p2X(BB): p2.Y = p2Y(BB)
        h1.X = h1X(BB): h1.Y = h1Y(BB): h2.X = h2X(BB): h2.Y = h2Y(BB)
        Poly_curves I
    Else
        Npt(I).X = (W(BB) - LeftM) * RightM:
        Npt(I).Y = (N(BB) - TopM) * BottomM: I = I + 1
    End If
Next BB
    API = SetPolyFillMode(Printer.hDC, 1): PolyPts = I - 1
        API = Polygon(Printer.hDC, Npt(1), PolyPts)
''''''''''''''''''''''''''''''''''''''''
    Case FlagBox(B)            'rectangle
        API = Rectangle(Printer.hDC, X1, Y1, X2, Y2)
    Case FlagBézier(B)      'Bézier curve
        If FlagBézier(B) = PolyShape(B) Then GoTo SkipCurve2
p1.X = p1X(B): p1.Y = p1Y(B): p2.X = p2X(B): p2.Y = p2Y(B)
h1.X = h1X(B): h1.Y = h1Y(B): h2.X = h2X(B): h2.Y = h2Y(B)
        Printer.ForeColor = OutLineColor(B)
        BezDraw
        SkipCurve2:
        Case FlagText(B) 'text
        Printer.FontSize = TextSize(B) * PageScale!
        Printer.FontName = TextFont(B)
        Printer.FontSize = TextSize(B) * PageScale!
        Printer.ForeColor = PaintColor(B)
        Printer.FontBold = TextBold(B)
        Printer.FontItalic = TextItalic(B)
        If FlagAlign(B) = TA_LEFT Then DT_FLAG = DT_LEFT
        If FlagAlign(B) = TA_CENTER Then DT_FLAG = DT_CENTER
        If FlagAlign(B) = TA_RIGHT Then DT_FLAG = DT_RIGHT
        Dim TextRect As Rect
        TextRect.Left = (W(B) - LeftM) * RightM
        TextRect.Top = (N(B) - TopM) * BottomM
        TextRect.Right = (E(B) - LeftM) * RightM
        TextRect.Bottom = (S(B) - TopM) * BottomM
        API = DrawText(Printer.hDC, StoredText(B), -1,
                        TextRect, DT_FLAG Or DT_NOCLIP)
End Select
'''''
    Printer.ForeColor = rr&: Printer.FillColor = 0: Printer.FillStyle = 1
    B = B + 1
Next Repeat
'''''''''''''''
        Printer.EndDoc
Next r
    FlagPrinter = False
    Screen.MousePointer = 0
    FlagPrinter = 0
    Exit Sub
PrintErr:
    FlagPrinter = False
    Screen.MousePointer = 0
    FlagPrinter = 0
```

```
        Printer. EndDoc
    Exit Sub
    End Sub
```

通过修改 RightM 和 BottomM 变量，可以以 600，1200 或更高的 DPI（点/英寸）比例向打印机输出图形。同时也要注意不是所有打印机都能打印 Windows GUI 函数，甚至 Visual Basic 图形语句。

APIH＝GetDeviceCaps(Printer. hDC，8)：得到以象素为单位的打印机水平分辨率。

APIV＝GetDeviceCaps(Printer. hDC，10)：得到以象素为单位的打印机垂直分辨率。

HPrinterDPI！＝APIH/(Pagex2－Pagex1)：计算打印机页面宽度与要打印的 Artwork 图片框控制对象的水平区域之间的误差。PageX1 和 PageX2 表示在 UpdateDesktop 过程（见源代码）中绘制在桌面上的页面图象的宽度。

VPrinterDPI！＝APIV/(PageY2－PageY1)：计算打印机页面高度与要打印的 Artwork 图片框控制对象的垂直区域之间的误差。PageY1 和 PageY2 表示在 UpdateDesktop 过程（见源代码）中绘制在桌面上的页面图象的高度。

RightM＝HPrinterDPI！：按比例增加或减少这些数值将会调节要打印的对象。

BottomM＝VPrinterDPI！：按比例增加或减少这些数值将会调节要打印的对象。

LeftM 和 BottomM 以窗体使用的当前 ScaleMode 来度量。为了手工调节打印在页面上的图象需要执行下列语句：

LeftM＝：图象的左侧页边距。修改这个数值将向左或向右移动打印机页上的所有图象。

TopM＝：图象的上侧页边距。修改这个数值将向上或向下移动打印机页上的所有图象。

RightM 和 BottomM 变量是两个设备使用的刻度之间的复合误差。

RightM＝：图象的右侧页边距。修改这个数值将水平增大或缩小打印机页上的所有图象。

BottomM＝：图象的下侧页边距。修改这个数值将垂直增大或缩小打印机页上所有图象。

15.41 本章小结

ArtAPI 程序显示了图形绘制过程的基本原理，这些过程把 Visual Basic 图象和图片框控制对象同 API 绘图函数结合了起来。大部分多边形过程都有一个简单而根本的节点创建结构。其中一些过程由于需要额外的代码行来重新定位 Visual Basic 图象和图片框控制对象（节点）而变得较长。现在希望做的可能是把 12 个多边形过程中的一部分合并为一个过程并把所有 Visual Basic 控制对象的定位操作移到一个单独的函数调用中。

创建直线或曲线节点的基本原则可分为以下几类：

一个节点即为一坐标点，可以连接以下几种情况之一：

连接节点 X 和 Y 坐标某侧的一条直线对象。

连接节点 X 和 Y 坐标某侧的一条曲线对象。

连接节点 X 和 Y 坐标某侧的一条直线和曲线对象。

连接对象起始或终止坐标点的一条直线或曲线对象。

因为 Visual Basic 的控制对象只有 Left，Top，Width 和 Height 位置属性，所以或者是在节点的 X 和 Y 点上加上偏移坐标值，或者是使用一个小函数来计算一个图象或图片框的中心点，然后相应地放置节点，如果应用程序只需要用节点来设计有限的折线或多边形结构，则可以把 Visual Basic 控制对象全部删去并用 PSet 语句来表示节点。这将大大减少应用程序所需的多边形过程和事件。许多二维和三维建筑学及实体游戏使用户可以用简单的 PSet 点来作为节点并定义多边形节点配置，同时用 Line 语句来构建楼层规划及内部设计。

Windows 16 位 API 迟早要被淘汰，应该考虑一下按照新的 GUI32 标准（见附录 B）来重建本书中的程序。在大多数情况下，只需简单地修改声明即可使程序升级并运行在 32 位操作环境中。如果读者从头到尾看了这本书，那么将会了解怎样手工创建大部分图形绘制例程。而如果集成了 GDI32 多边形和路径函数，则一部分例程就不再需要了。尽管如此，还得创建数组并手工编制可增强某些函数功能的绘图例程。一些明显可以替代本书中手工编程计算的 GDI32 函数是 PolyBézier 和 PolyBézierTo 函数。但是仍需要编写例程来创建 Bézier 图柄和节点。

我希望本书将使你了解图形编程的原理，并进而把你的程序提到新的视觉高度上。

附录 A Artisan.BAS 文件

A.1 ART3_7.BAS 文件

1 附录 A 包括 Artisan 程序（第 2 章到第 5 章）和 ArtAPI 程序（第 15 章）所用的变量、常数及过程的全局声明，ART3_7.BAS 文件和 ARTAPI.BAS 文件基本相同，前者是 Artisan 程序的主模块文件，后者是 ArtAPI 程序的主模块文件。

A.2 ART3_7.BAS 文件中的全局声明

清单 A.1 列出了 Artisan 和 ArtAPI 程序中所有窗体使用的全局变量。另外，本清单中还列出了两个全局过程，分别是 Display_Color_Palette 和 Update_Mouse_Colors。这两个过程是 Visual Basic 的 Icon Works（图标制作器）中过程的改进版，并被用于 Artisan 和 ArtAPI 程序来创建调色板并选取颜色。

附录 A 还包括用于 Artisan 方案中 Text 窗体的 APIFONT.BAS 全局变量清单。因为此文件同时也适用于其他程序，所以在编辑之前最好将它换个名字保存起来。

清单 A.1 Artisan 程序中的全局声明

```
DefInt A-Z

'Text Labels
Global Align As Integer
Global TextPoint As Single
Global CarriageCount(1000) As Integer
Global crCounter As Integer

'Zoom Bar Flags
Global Zoom_In， Zoom_Out As Integer

' Bezier Curves
Global Convert_Curve%
Global Convert_Line%
Global FlagPrinter%

Type POINTS
    X As Integer
    Y As Integer
End Type

Type Rect
       Left As Integer
       Top As Integer
       Right As Integer
       Bottom As Integer
End Type
''''
Declare Sub GetCursorPos Lib "User" (lpPoint As POINTAPI)
Declare Function GetMenu Lib "User" (ByVal hWnd)
Declare Function GetPrivateProfileString Lib "Kernel" (ByVal Appname$,
    ByVal KeyName$, ByVal DEFAULT$, ByVal ReturnedString$,
    ByVal MaxSize, ByVal FileName$)
Declare Function GetSubMenu Lib "User" (ByVal hWnd, ByVal Position)
Declare Function SendMessageByNum% Lib "User" Alias "PostMessage" (ByVal
    hWnd%, ByVal wMsg%, ByVal wParam%, ByVal lParam&)
Declare Function TrackPopupMenu Lib "User" (ByVal hMenu, ByVal E2,
    ByVal X, ByVal Y, ByVal r2, ByVal hWnd, ByVal r3&)
Declare Function WritePrivateProfileString Lib "Kernel" (ByVal Appname$,
    ByVal KeyName$, ByVal NewString$, ByVal FileName$)
''''''''''''''''

'odds & ends
Global Const SRCCOPY = &HCC0020
Global Const SRCAND = &H8800C6
Global Const SRCINVERT = &H660046
Global Const Blackness = &H42&
Global Const MF_BYPOSITION = &H400
Global Const BITSPIXEL = 12
Global Const HELP_CONTEXT = &H1
Global Const HELP_QUIT = &H2
Global Const HELP_HELPONHELP = &H4

'KeyWord Names
Global Const KEY_NAME = "Artisan"
Global Const KEY_ENTRY = "Colors Row"
Global Const INI_FILENAME = "Artisan.INI"

'Default Color palette values
Global Const BAS_PALETTE1 = "16777215 14737632 12632319 12640511 14745599
    12648384 16777152 16761024 16761087 00000192
    00016576 00049344 00049152 12632064 12582912 12583104"
Global Const BAS_PALETTE2 = "12632256 04210752 08421631 08438015 08454143
    08454016 16777088 16744576 16744703 00000128
    00016512 00032896 00032768 08421376 08388608 08388736"
Global Const BAS_PALETTE3 = "08421504 00000000 00000255 00033023 00065535
    00065280 16776960 16711680 16711935 00000064
    04210816 00016448 00016384 04210688 04194304 04194368"

'Default value for Artisan Filename field
Global Const UNTITLED = "[Untitled]"

'Artisan's ToolPalette ID's
Global Const PointerTool = 0
Global Const BezierTool = 1
Global Const ZoomTool = 2
Global Const LineTool = 3
Global Const BoxTool = 4
Global Const CircleTool = 5
Global Const OutLineTool = 6
Global Const PaintTool = 7
Global Const TextTool = 8
Global Const Poly_line = 9

'Artisan's ZoomBar ID's
Global Const ZoomIn = 0
Global Const ZoomOut = 1
Global Const ZoomAll = 2
Global Const ZoomPage = 3
'Artisan's ToolLine ID's
Global Const LineColor = 0
Global Const NoLine = 1
Global Const HairLine = 2
Global Const OnePt = 3
Global Const TwoPt = 4
Global Const FourPt = 5
Global Const EightPt = 6
Global Const TwelvePt = 7
'Artisan ToolFill ID's
Global Const Palette = 0
Global Const NoFill = 1
Global Const WhiteBrush = 2
Global Const Brush10 = 3
Global Const Brush20 = 4
Global Const Brush50 = 5
Global Const Brush70 = 6
Global Const BlackBrush = 7
'Index's into the ColorPalettes' Scrl_RGB()
Global Const RED_ELEMENT = 0
Global Const GREEN_ELEMENT = 1
Global Const BLUE_ELEMENT = 2
' Color palette array
Global Colors(47) As Long
Global PenColor As Long
Global BrushColor As Long
'Artisan global Variables
Global Opaque%
Global CurrentTool%
Global LastToolUsed%
Global CurrentLine%
Global CurrentBrush%
Global CurrentZoom%
Global ColorIndex%
Global ColorString      As String
' Miscellaneous
Global Text                 As String
Global CRLF                  As String * 2
Global A_TAB                  As String * 1
' WindowState
Global Const MINIMIZED = 1
' Clipboard formats
Global Const CF_BITMAP = 2
' Show parameters
Global Const MODAL = 1
Global Const MODELESS = 0
' Colors
Global Const BLACK = &H0&
Global Const DARKBLUE = &H800000
Global Const DARKGREEN = &H8000&
Global Const DARKCYAN = &H808000
Global Const DARKRED = &H80&
Global Const DARKPURPLE = &H800080
Global Const DARKYELLOW = &H8080&
Global Const DARKGRAY = &H808080
Global Const GRAY = &HC0C0C0
Global Const BLUE = &HFF0000
Global Const GREEN = &HFF00&
Global Const CYAN = &HFFFF00
Global Const RED = &HFF&
Global Const PURPLE = &HFF00FF
Global Const YELLOW = &HFFFF&
Global Const WHITE = &HFFFFFF
' MousePointer
Global Const DEFAULT = 0
Global Const CROSSHAIR = 2
Global Const SIZE = 5
Global Const UP_ARROW = 10
Global Const HOURGLASS = 11
' DrawMode Values
Global Const Black_Pen = 1
Global Const INVERSE = 6
Global Const Xor_Pen = 7
Global Const COPY_PEN = 13
' DrawStyle property Values
Global Const SOLID = 0
Global Const DOT = 2
' FillStyle Values
Global Const TRANSPARENT = 1
' Key Codes
Global Const KEY_BACK = &H8
Global Const KEY_SHIFT = &H10
Global Const KEY_CONTROL = &H11
Global Const KEY_INSERT = &H2D
Global Const KEY_DELETE = &H2E
Global Const KEY_SPACE = &H20
Global Const KEY_F1 = &H70
' Shift parameter masks
Global Const SHIFT_MASK = 1
Global Const CTRL_MASK = 2
Global Const ALT_MASK = 4
' Button parameter masks
Global Const LEFT_BUTTON = 1
Global Const RIGHT_BUTTON = 2
Global Const MIDDLE_BUTTON = 4
' ScaleModes
Global Const TWIPS = 1
Global Const PIXELS = 3
' MsgBox return values
Global Const MBYES = 6
Global Const MBNO = 7
' PageSize
Global PageX1 As Integer
Global PageY1 As Integer
Global PageX2 As Integer
Global PageY2 As Integer
```

ARTFONT.BAS 文件

```
Type FontLog
    FontHeight As Integer
    FontWidth As Integer
    FontRotate As Integer
    FontOrientation As Integer
    FontWeight As Integer
    FontItalic As String * 1
    FontUnderLine As String * 1
    FontStrikeThru As String * 1
    FontCharSet As String * 1
    FontOutPrecision As String * 1
    FontClipPrecision As String * 1
    FontQuality As String * 1
    FontPitchFamily As String * 1
    FontFaceName As String * 20
End Type
''''''''
Type TEXTMETRIC
    tmHeight As Integer
    tmAscent As Integer
    tmDescent As Integer
    tmInternalLeading As Integer
    tmExternalLeading As Integer
    tmAveCharWidth As Integer
    tmMaxCharWidth As Integer
    tmWeight As Integer
    tmItalic As String * 1
    tmUnderlined As String * 1
    tmStruckOut As String * 1
    tmFirstChar As String * 1
    tmLastChar As String * 1
    tmDefaultChar As String * 1
    tmBreakChar As String * 1
    tmPitchAndFamily As String * 1
    tmCharSet As String * 1
    tmOverhang As Integer
    tmDigitizedAspectX As Integer
    tmDigitizedAspectY As Integer
End Type
''''''''
Type POINTAPI
    x As Integer
    y As Integer
End Type
''''''''
Type GLYPHMETRICS
    gmBlackBoxX As Integer
    gmBlackBoxY As Integer
    gmptGlyphOrigin As POINTAPI
    gmCellIncX As Integer
    gmCellIncY As Integer
End Type
''''''''

Type Fixed
    fract As Integer
    value As Integer
End Type
''''''''

Type Mat2
    eM11 As Fixed
    eM12 As Fixed
    eM21 As Fixed
    eM22 As Fixed
End Type
''''''''

Type POINTFX
    x As Fixed
    y As Fixed
End Type
''''''''

Type TTPOLYGONHEADER
    cb As Long
    dw As Long
    pfxStart As POINTFX
End Type
''''''''

Type TTPOLYCURVE
    wType As Integer
    cpfx As Integer
    pfx(0 To 2) As POINTFX
End Type
''''''''

Declare Function CreateFontIndirect% Lib "GDI" (lpFontLog As FontLog)
Declare Function DeleteObject% Lib "GDI" (ByVal hObject%)
Declare Function GetTextMetrics% Lib "GDI" (ByVal hDC%,
    lpMetrics As TEXTMETRIC)
Declare Function GetTextFace% Lib "GDI" (ByVal hDC%, ByVal nCount%,
    ByVal lpFacename$)
Declare Function GetGlyphOutLine& Lib "GDI" (ByVal hDC%, ByVal uChar%,
    ByVal fuFormat%, lpgm As GLYPHMETRICS, ByVal cbBuffer&,
    lppt As POINTAPI, lpmat2 As Mat2)
Declare Function GlobalUnlock% Lib "Kernel" (ByVal hMem%)
Declare Function GlobalFree% Lib "Kernel" (ByVal hMem%)
Declare Function PolyPolygon% Lib "GDI" (ByVal hDC%, lpPoints As POINTAPI,
    lpPolyCounts%, ByVal nCount%)
Declare Function SelectObject% Lib "GDI" (ByVal hDC%, ByVal hObject%)
Declare Function SetTextAlign% Lib "GDI" (ByVal hDC%, ByVal wFlags%)
Declare Function TextOut% Lib "GDI" (ByVal hDC%, ByVal x%, ByVal y%, ByVal
    lpString$, ByVal nCount%)
'''''''''''''''''

Global FontSelection%
'''''''''''''''''

Global Const ANSI_CHARSET = 0
Global Const CLIP_CHARACTER_PRECIS = 1
Global Const CLIP_DEFAULT_PRECIS = 0
Global Const CLIP_EMBEDDED = &H80
Global Const CLIP_LH_ANGLES = &H10
Global Const CLIP_STROKE_PRECIS = 2
Global Const CLIP_TT_ALWAYS = &H20

Global Const DEFAULT_QUALITY = 0
Global Const DEFAULT_CHARSET = 1
Global Const DEFAULT_PITCH = 0

Global Const DEVICE_FONTTYPE = 2
Global Const DRAFT_QUALITY = 1

Global Const DT_TOP = &H0
Global Const DT_LEFT = &H0

Global Const DT_BOTTOM = &H8
Global Const DT_CALCRECT = &H400
Global Const DT_CENTER = &H1
Global Const DT_EXPANDTABS = &H40
Global Const DT_EXTERNALLEADING = &H200
Global Const DT_INTERNAL = &H1000
Global Const DT_NOCLIP = &H100
Global Const DT_NOPREFIX = &H800
Global Const DT_RIGHT = &H2
Global Const DT_SINGLELINE = &H20
Global Const DT_TABSTOP = &H80
Global Const DT_VCENTER = &H4
Global Const DT_WORDBREAK = &H10

Global Const FIXED_PITCH = 1

Global Const FF_DECORATIVE = 80
Global Const FF_DONTCARE = 0
Global Const FF_MODERN = 48
Global Const FF_ROMAN = 16
Global Const FF_SCRIPT = 64
Global Const FF_SWISS = 32

Global Const FW_BOLD = 700
Global Const FW_DONTCARE = 0
Global Const FW_EXTRABOLD = 800
Global Const FW_EXTRALIGHT = 200
Global Const FW_HEAVY = 900
Global Const FW_LIGHT = 300
Global Const FW_MEDIUM = 500
Global Const FW_NORMAL = 400
Global Const FW_SEMIBOLD = 600
Global Const FW_THIN = 100

Global Const LF_FULLFACESIZE = 64

Global Const NTM_BOLD = &H20&
Global Const NTM_ITALIC = &H1&
Global Const NTM_REGULAR = &H40&

Global Const OEM_CHARSET = 255

Global Const OUT_CHARACTER_PRECIS = 2
Global Const OUT_DEFAULT_PRECIS = 0
Global Const OUT_DEVICE_PRECIS = 5
Global Const OUT_RASTER_PRECIS = 6
Global Const OUT_STRING_PRECIS = 1
Global Const OUT_STROKE_PRECIS = 3
Global Const OUT_TT_ONLY_PRECIS = 7
Global Const OUT_TT_PRECIS = 4

Global Const PROOF_QUALITY = 2

Global Const RASTER_FONTTYPE = 1

Global Const SHIFTJIS_CHARSET = 128

Global Const SYMBOL_CHARSET = 2

Global Const TA_CENTER = 6
Global Const TA_BOTTOM = 8
Global Const TA_BASELINE = 24
Global Const TA_LEFT = 0
Global Const TA_NOUPDATECP = 0
Global Const TA_RIGHT = 2
Global Const TA_TOP = 0
Global Const TA_UPDATECP = 1

Global Const TMPF_DEVICE = 8
Global Const TMPF_FIXED_PITCH = 1
Global Const TMPF_TRUETYPE = 4
Global Const TMPF_VECTOR = 2

Global Const TRUETYPE_FONTTYPE = 4

Global Const VARIABLE_PITCH = 2
```

A.3　给调色板赋以颜色

清单 A.2 中的 Display_Color_Palette 过程用于 Artisan 主窗体和彩色调色板窗体，其功能是绘制 ColorBar 图片框控制对象中的每个颜色格及分割线(ColorBar 是位于 Artisan 和 ArtAPI 主窗体底部的主调色板)。

清单 A.2　Artisan 调色板的颜色值

```
Sub Display_Color_Palette (ColorBar As Control)
    ColorBar.Scale (0, 0)-(48, 1)

    For I = 0 To 47
        ColorBar.Line (I, 0)-(I + 1, 1), Colors(I), BF
        If I Then ColorBar.Line (I, 0)-(I, 3)
    Next I
End Sub
```

ColorBar.Scale：使用自定义刻度将图片框控制对象分为 48 个小区域。高度被定为一个单位，这样不管对 ColorBar 施行什么操作，48 个颜色块总是按着控制对象的长度排成一行。

For I= 0 To 47：对 0 到 47 的每个颜色，做如下工作。

ColorBar.Line：一次一个画出调色板颜色项，循环一次画一个。

Color(I)：Color 是保存颜色值的数值，索引号为从 0 到 47。

If I Then ColorBar. Line：在每两色间画条线以求更好的视觉效果。

注意此句中高度被写为三个单位。这是因为主ColorPalette窗体(编辑调色板)也用到了Display_Color_Palette过程，但显示颜色的图片框是三个颜色行的高度。

A.4　给对象赋以颜色

清单A.3列出Update_Mouse_Colors。当鼠标左键点中调色板中某颜色时，即为对象赋以该填充色，当用鼠标右键点中调色板中颜色时，为对象赋以轮廓色。

清单A.3　单击鼠标并赋以颜色

```
Sub Update_Mouse_Colors (Button, X As Single, Y As Single)
Dim color As Long
    ColorIndex = Fix(X) + Fix(Y) * 47
color = Colors(ColorIndex)
If Button = RIGHT_BUTTON Or CurrentTool = 6 Then
PenColor = color
Artisan. StatusColor. BackColor = color
Exit Sub
End If
BrushColor = color
Artisan. StatusPaint. BackColor = color
End Sub
```

Dim Color：颜色值，一个Long型16位值。

ColorIndex：返回MouseDown事件的CurrentX和CuurentY位置，同时也表示了得到的颜色索引号。X和Y是十进制形式，因此用FIX方法来取整。

If Button = RIGHTBUTTON：当击下鼠标右键时设置对象的轮廓色。

ArtisanStatusColor：这是位于窗体右下角的小图片框，用作颜色指示器。使用BackColor属性而非BorderColor属性是为了看到抖动色。

BrushColor：当击下鼠标左键时设置对象的填充色。

Artisan. StatusPaintColor：这是位于StatusColor图片框中的小型颜色指示器，代表对象的当前填充色。使用较小的图片框控制对象是为了看到位于其后的代表对象轮廓色的StatusColor图形框。

A.5　Artisan1 Text窗体

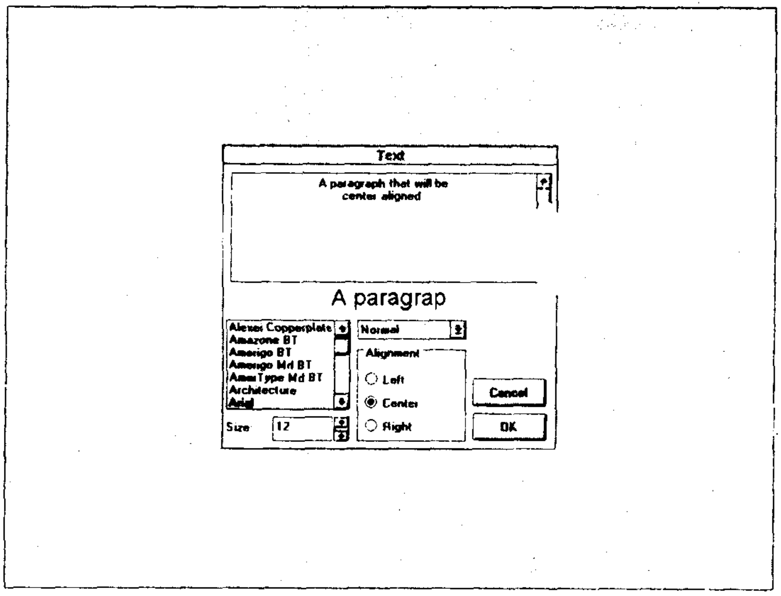
Text窗体使用基本的文本放置方法而不用API函数，虽然API文本放置函数也许更好一些。不过，对于专业文字处理来说，最好先彻底了解True Type字体技术，以免发现改动本窗体是多么复杂。

本节中的控制对象及代码都很易懂。ReDrawText过程从事主Artisan窗体中实际的文字绘制工作。

在运行时改变文本框控制对象的对齐方式属性是不允许的。窗体中的文本框控制对象

实际上是由三个重叠的文本框控制对象组成的。当输入文本时，具有正确对齐方式的文本框被显示，其余两个被掩藏起来。

使用 Visual Basic 的 Print 语句放置文本时的难点就是如何在使用缩放工具时适当地调整文本点阵大小。Visual Basic 一般对小于 8 点阵的规格化字体自动进行取整。鉴于 Visual Basic 使用一个位图字形来模拟小字体，必须调用两次字体尺寸或字体名属性。这就给用十进制定义的点阵大小造成了混乱（如：4.37 点阵的文本）。由于以上原因，没有为改进 Artisan 程序的缩放工具做更多尝试，缩小或放大模式中文字的大小仅是一近似值。因此，某些文本将不会以真正的 WYSIWYG（所见即所得）点阵大小显示。图 A.1 显示了 Text 窗体的外表。


Text
Alexei Copperplate
Amazone BT
Amerigo BT
Amerigo Md BT
AmerType Md BT
Architecture
Arial
Size
Alignment
○ Left
○ Right
Cancel
OK

A paragraph that will be
center aligned

A paragrap

Size 12

Normal

● Center

图 A.1　Artisan 程序的文字编辑窗体

清单 A.4 的声明中包括了 Text 窗体的初始变量。

清单 A.4　调入窗体时使用的变量

Declaration

```
Dim Z As Integer
Dim LastFontSize As Integer
Dim LastFontName As Strint
```

Dim Z：保存当前文本框索引号。

Dim LastFontSize：如果用户取消了对文字大小的改动，则保存当前字体大小。

Dim LastFontName：如果用户取消了改动字体操作，则保存当前字体名。

清单 A.5 中的 Form_Load 事件初始化用于 Text 窗体的默认设置。

清单 A.5 设置默认值

```
Sub Form_Load ()
TextForm. Move (Screen. Width－TextForm. Width) / 2,
            (Screen. Height－TextForm. Height) / 2
    Style_Combo. AddItem "Normal"
    Style_Combo. AddItem "Bold"
    Style_Combo. AddItem "Italic"
    Style_Combo. AddItem "Bold. Italic"

Text1(1). Visible = False
Text1(2). Visible = False
    Sizebox. Text = ARTISAN. ArtWork. FontSize
    LastFontName = ARTISAN. ArtWork. FontName
    LastFontSize = Sizebox. Text
    ViewFont. FontSize = 18

For R = 0 To 20 'Printer. FontCount－1
    FontNames $ = Printer. Fonts(R)
    FontList. AddItem FontNames $
Next R

For i = 0 To FontList. ListCount － 1
    If FontList. List(i) = ARTISAN. ArtWork. FontName Then FontList. ListIndex = i
Next i

    Style_Combo. ListIndex = 0
    crCounter = 0

End Sub
```

TextForm. Move：将文本窗体置于屏幕中央。

Style_Combo. AddItem"Normal"：向组合框列表中添加字体属性名称。

Text(1). Visible＝False：隐藏居中方式和右对齐方式文本框。

Sizebox. Text：使程序最后使用的字体大小成为当前字体大小。

LastFontName：如果用户取消了对文本的改动，则保存当前字体名。

LastFontSize：如果用户取消了对文本的改动，则保存当前字体尺寸。

ViewFont. FontSize：在标题控制对象中预显示字体。

For R＝0 To 20 'Printer. FontCount－1：为了节省时间，仅调入 20 种打印字体。如果你的系统中安装的字体少于 20 种的话，就有必要改变这个值。如果不想使用打印机，那么把 Printer. FontCount－1 或 Screen. FontCount－1，就会显示系统中所有打印字体或屏幕字体。

FontList. AddItem FontNames $：把字体名加到列表框控制对象里。

For i＝0 To FontList. ListCount－1：查找列表框中所有字体名，并将程序当前字体高亮显示。

Style_Combo. ListIndex＝0：将字体属性置为正常字形。

crCounter＝0：程序要保存你键入的回车数，并用于在使用 Print # 或 Write # I/O 文件方法保存或打开文件时提取文本行。

如果用户取消了文本编辑操作，清单 A.6 中的 Cancel_Btn_Click 事件重置初始值。

清单 A. 6　取消文本编辑函数的操作

```
Sub Cancel_Btn_Click ()
    ARTISAN. ArtWork. FontName = LastFontName
    ARTISAN. ArtWork. FontSize = LastFontSize
    ARTISAN. ArtWork. FontBold = False
    ARTISAN. ArtWork. FontItalic = False
    ARTISAN. Txt. FontName = LastFontName
    ARTISAN. Txt. FontSize = LastFontSize
    ARTISAN. Txt. FontBold = False
    ARTISAN. Txt. FontItalic = False
    ARTISAN. Txt. Width = 15
    Align = 0

    Unload TextForm

End Sub
```

ARTISAN. ArtWork. FontName：选择 Cancel(取消)按钮后，所有文本属性被重置为默认设置。

Unload TextForm：卸去文本窗体。因为更新代码在 Form_Load 事件中(见表单 A. 5)，简单地隐藏窗体不会自动更新属性。

清单 A. 7 中的 CarriageReturns 过程统计用户按下的回车键数目。向文件保存文字串时需要回车符标志。

清单 A. 7　计算字串中回车数

```
Sub CarriageReturns ()
    Static curindex As Long
    Dim Find As String
    Find = Chr(13)

Do
     curindex = InStr(curindex + 1, Text1(Z). Text, Find)
     If curindex > 0 Then

     Text1(Z). SelStart = curindex . 1
     Text1(Z). SelLength = Len(Find)
     CR_LF = CR_LF + 1
  End If

Loop While curindex > 0

    crCounter = CR_LF
End Sub
```

Find_Chr(13)：程序保存你键入的回车数目，但仅用于在使用 Print # 或 Write # I/O 文件方法保存或打开文件时提取文本行。这里用的方法与第 7 章中多重查找和替换程序里的例程完全一样。请参考第 7 章来详细了解清单 A. 7 中的代码。

清单 A. 8 中的 FontList_Click 事件是用来从列表框中选择一个新字体的。

清单 A.8　从列表框中选择一个新字体

```
Sub FontList_Click ()
    ViewFont.FontName = FontList.Text
    ARTISAN.ArtWork.FontName = ViewFont.FontName
    ARTISAN.Txt.FontName = ViewFont.FontName
    ARTISAN.Txt.Width = 15
End Sub
```

ViewFont. FontName：已从列表框中选择了一个新字体名，相应地改变所有字体名属性。Artisan. Txt 标题控制对象被设为可自动调节大小，这样当字体名改变时会自动扩展。Txt 标题用作光标，所以宽度设为 15 twip，即一个象素宽。

当用户选择 Text 窗体的 OK 按钮时，清单 A.9 中的 OK_Btn_Click 事件接受对文本的改动。

清单 A.9　在桌面上打印编辑过的文本

```
Sub OK_Btn_Click ()
If Text1(Z).Text > "" Then 'print attributes
    ARTISAN.ArtWork.FontName = ViewFont.FontName
    ARTISAN.ArtWork.FontBold = ViewFont.FontBold
    ARTISAN.ArtWork.FontItalic = ViewFont.FontItalic
ARTISAN.Txt.Visible = False
ARTISAN.Txt.FontName = ViewFont.FontName
ARTISAN.Txt.FontSize = Val(Sizebox.Text)
If Zoom_In = True Then ARTISAN.Txt.FontSize = ARTISAN.Txt.FontSize * TextPoint
If Zoom_Out = True Then ARTISAN.Txt.FontSize = ARTISAN.Txt.FontSize * TextPoint
ARTISAN.Txt.Caption = Text1(Z).Text
ARTISAN.StatusLabel.Caption = ViewFont.FontName
    Select Case True
        Case Option1(0)
            Align = 0 'left
        Case Option1(1)
            Align = 6 'center
        Case Option1(2)
            Align = 2 'right
    End Select
End If
        Unload TextForm
End Sub
```

If Text1(Z). Text>"" Then：选择了 OK 按钮。如果文本框中已有输入字符，则在桌面上重绘。如果文本框中没有字符，则任何属性的改变都将影响桌面上新的输入。

清单 A.10 中的 Option1_Click 事件改变文本的对齐方式(左对齐、居中或右对齐之一)。

清单 A.10　对齐文本框中的文本

```
Sub Option1_Click (Index As Integer)
Select Case Index
     Case 0 'Align Left
Text1(0).Text = Text1(Z).Text
Text1(0).Visible = True: Z = Index: Text1(0).SetFocus
Text1(1).Visible = False: Text1(2).Visible = False
     Case 1 'Align Centre
Text1(1).Text = Text1(Z).Text
Text1(1).Visible = True: Z = Index: Text1(1).SetFocus
Text1(0).Visible = False: Text1(2).Visible = False
     Case 2 'Align Right
Text1(2).Text = Text1(Z).Text
Text1(2).Visible = True: Z = Index: Text1(2).SetFocus
Text1(0).Visible = False: Text1(1).Visible = False
End Select
End Sub
```

Select Case Index：根据你选择的对齐方式选项，执行后续工作。

Text1(0): Text=Text1(Z).Text：把文本窗体上某对齐方式的文本框切换至另一个。

Text1(0).Visible=True;Z=Index：显示新的对齐方式的文本框，并保存索引号。

Text1(1).Visible=False：隐藏旧的文本框。

清单 A.11 中的 SizeBox_Change 事件改变字体点阵大小。

清单 A.11　手工输入一个新的字体尺寸

```
Sub SizeBox_Change ()
On Error GoTo BadInput
     Pts = Val(Sizebox.Text)
     Pt2 = Fix(Pts)
     VScroll1.Value = Pt2: Exit Sub
BadInput:
     Sizebox.Text = "12"
     Exit Sub
End Sub
```

On Error GoTo BadInput：可以在文本框中输入一个字体尺寸，此尺寸应介于 VScroll1 滚动条控制对象的最小值和最大值之间。如果你输入一个字母或输入值超出了最小值与最大值的范围，字体尺寸默认设为 12。

清单 A.12 中的 Style_Combo_Click 事件修改字体的正常、斜体及粗体属性。

清单 A.12 选择新的字体属性

```
Sub Style_Combo_Click ()
Select Case Style_Combo.Text
    Case "Normal"
        ViewFont.FontBold = False: ViewFont.FontItalic = False
    Case "Bold"
        ViewFont.FontBold = True: ViewFont.FontItalic = False
    Case "Italic"
        ViewFont.FontItalic = True: ViewFont.FontBold = False
    Case "Bold-Italic"
        ViewFont.FontBold = True: ViewFont.FontItalic = True
End Select
End Sub
```

Select Case Style_Combo.Text：根据所选择的组合框条目改变字体属性。

清单 A.13 中的 Text1_Change 事件接收用户输入文本并按指定格式显示前 10 个字符。

清单 A.13 显示示例文本

```
Sub Text1_Change (Index As Integer)
Z = Index: ViewFont.Caption = Left$ (Text1(Z).Text, 10)
End Sub
```

Z＝Index：ViewFont.Caption：如果在文本框中有输入文本，则在预显示标题控制对象中显示前 10 个字符。

如果用户使用了垂直滚动条控制对象，则清单 A.14 中 VScroll_Change 事件改变当前字体大小。

清单 A.14 使用滚动条控制对象改变字体大小

```
Sub VScroll1_Change ()
    Sizebox.Text = Format$ (VScroll1.Value)
    ARTISAN.ArtWork.FontSize = Sizebox.Text
    ARTISAN.Txt.Visible = False
    ARTISAN.Txt.FontSize = Sizebox.Text
    ARTISAN.Txt.Width = 15
    ARTISAN.Txt.Visible = True
End Sub
```

Sizebox.Text＝Format$ (VScroll1.Value)：在字体大小文本框中显示当前字体大小的变化，并修改程序正在使用的所有字体大小。

A.6 Text 窗体的总结

Artisan 程序的文本编辑窗体具有有限文本特性，但是修改它非常容易。为了显示和编

辑桌面上已有的文本，可以使 StoredText(B)数组成为全局数组，具体做法是将数组声明放入两个模块文件之一。所有字串都存在 StoredText 数组中，B 表示赋给字串的对象编号。可以在 Form_Load 事件中加入代码以决定桌面上当前选定的对象是否有 FlagText 属性。如果有的话，就把与指定编号的对象相匹配的 StoredText 串置入文本编辑框。

A.7　Artisan NodeEdit 窗体

NodeEdit 窗体是一个简单对话框，用于把直线转换为曲线，或把曲线转换为直线。全局变量 Convert_Curve 和 Convert_Line 用作 Artwork_DblClick 事件的标志并决定作何种转换。图 A.2 显示了 NodeEdit 窗体。

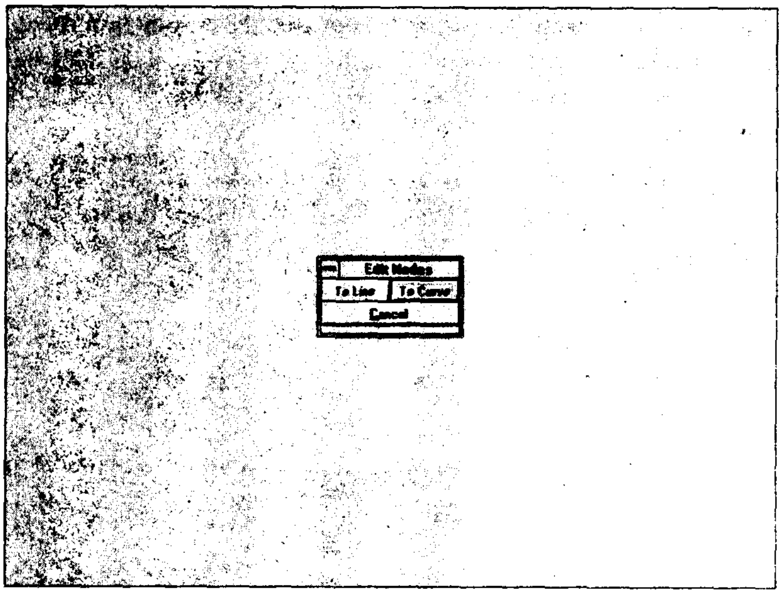


图 A.2　Artisan 程序的节点编辑窗体

如果选择了 Cancel 按钮，清单 A.15 中 Cancel_Click 事件将隐藏 NodeEdit 窗体。

清单 A.15　按下 Cancel(取消)按钮

```
Sub Cancel_Click ()
    NodeEdit.Hide
End Sub
```

NodeEdit.Hide：取消转换操作并隐藏节点编辑窗体。

如果选择了 To Curve 按钮，To_Curve_Click 事件将通知 Artwork_DblClick 事件把直线转换为曲线。

```
Sub To_Curve_Click()
Convert_Curve=True:NodeEdit.Hide
End Sub
```

Convert_Curve=True：选择了“转换为曲线”按钮，将使 Artwork 双击鼠标事件把直线

对象转换为 Bézier 曲线。

如果选择了 To Line 按钮，To_Line_Click 事件通知 Artwork_DblClick 事件把曲线变为直线。

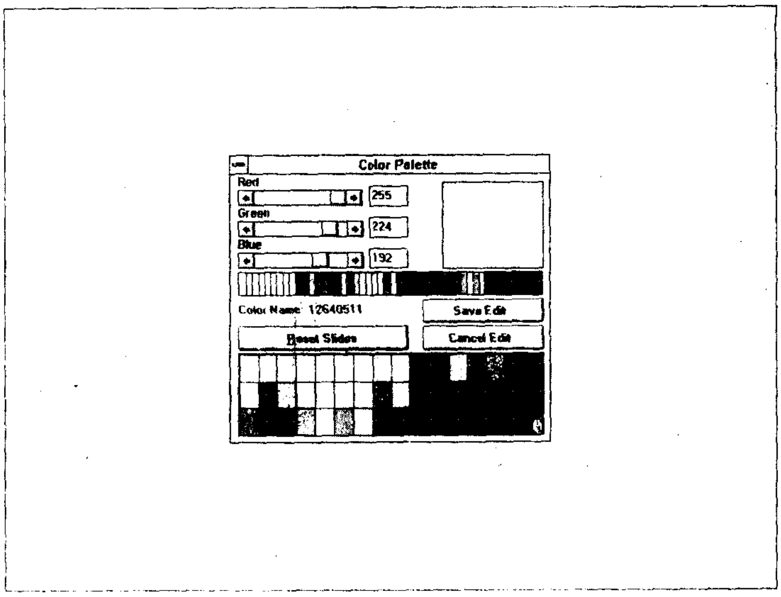
```
Sub To_Line_Click()
Convert_Line=True;NodeEdit.Hide
End Sub
```

Convert_Line=True：选择了“转换为直线”按钮，将使 Artwork_DblClick 事件把 Bezier 曲线转换为直线对象。

A.8 Artisan ColorPalette(彩色调色板)窗体

ColorPalette 窗体是一个很简单地调色板编辑器。它具有一些附加特性，如拖放式颜色修改。你可以用 RGB 滚动条来创建一个自定义颜色，然后将此新颜色拖至主调色板显示区。对第二调色板的任何修改都将保存到文件 ARTISAN. INI。

在修改了缺省彩色调色板之后，本程序在每次启动时使用 ARTISAN. INI 调色板信息。为了让 Artisan 程序在启动时使用缺省调色板，应删去 ARTISAN. INI 文件。图 A. 3 显示了调色板窗体。



Red 255
Green 224
Blue 192
Color Name: 12640511

图 A. 3 Artisan 程序的彩色调色板窗体

A.9 调色板窗体使用的值

调色板窗体是一个简单的编辑器，可用来修改 Artisan 程序中的可用颜色。只要拖放一个新定义的颜色到调色板显示区即可。清单 A. 16 中的 Form_Load 事件将调色板窗体置于屏幕中央，然后将当前彩色调色板调入调色板编辑图形框。

清单 A. 16　启动时的默认值

```
Sub Form_Load ()
        ColorPalette. Left = (Screen. Width－Width)/2
        ColorPalette. Top = (Screen. Height－Height)/2

    BrushColor = Artisan. StatusPaint. BackColor
    RGBscroll(0). Value = BrushColor And & HFF&
    RGBscroll(1). Value = (BrushColor And & HFF00&)\256
    RGBscroll(2). Value = (BrushColor And & HFF0000)\65536

        For i= 0 To 47
            paletteColors(i) = Colors(i)
        Next i

End Sub
```

ColorPalette. Left：将窗体置于屏幕中央。

BrushColor＝Artisan. StatusPaint. BackColor：桌面上当前选中对象的填充色。

RGBscroll(0). Value：**RGB** 颜色的 16 进制红色分量。

RGBscroll(1). Value：**RGB** 颜色的 16 进制绿色分量。

RGBscroll(2). Value：**RGB** 颜色的 16 进制蓝色分量。

For i＝0 To 47：保存 Artisan 当前的 48 色调色板。

PaletteColors(i)＝Colors(i)：每个 Artisan 调色板颜色都存于数组 Colors(i)中，所以将当前调色板值临时存储于数组 PaletteColors(i)中。

当选中 OK 按钮时，清单 A. 17 中的 OKbtn_Click 事件接收对彩色调色板的编辑修改。

清单 A. 17　保存或取消对调色板颜色的编辑

```
Sub OKbtn_Click ()
    If Index = 0 Then
        For i = 0 To 47
            Colors(i)=PaletteColors(i)
        Next i
    End If

    Unload ColorPalette

End Sub
```

Colors(i)＝PaletteColors(i)：选中“保存编辑(Save Edit)”按钮，故为 Artisan 的所有 48 个调色板颜色重新赋以新值。

Unload ColorPalette：选中“保存编辑(Save Edit)”或“取消编辑(Cancal Edit)”按钮，故卸去窗体。

清单 A. 18 中的 Cmd_Reset_Click 事件将 RGB 滚动条重置为编辑前的颜色值。

清单 A. 18　重置滚动条为前一次的值

```
Sub Cmd_Reset_Click ()
```

```
    GetRGBcolors
End Sub
```

GetRGBColors：将彩色分量值重置为初始设置。

清单 A.19 中的 ColorBar_MouseUp 事件把选中的颜色从小调色板中移到颜色显示图形框中，为下一步的编辑做准备。

清单 A.19 从 Artisan 程序的调色板中选取颜色

```
Sub ColorBar_MouseUp(Button As Integer,Shift As Integer, X As Single,Y As Single)
Dim Color As Long
X = Fix(X):Y = Fix(Y)
ColorIndex = X + Y * 47
    If X>=0 And X<48 And Y>=0 And Y<2 Then
        color =Colors(ColorIndex)
        GetRGBcolors
        ColorName.Caption = "Color Name: "+Format $ (DragColor.BackColor)
    End If
En Sub
```

Dim color As Long：所有颜色值均以长型 16 进制值存储。

X=Fix(X):Y=Fix(Y)：ColorBar 图形框被设为 48 个单位宽，1 个单位高。MouseUp 的 X 和 Y 参数是 Single 型的（十进制），Fix 语句的作用是将鼠标 X，Y 坐标截成整数。

ColorIndex=X+Y * 47：Artisan 调色板共有 48 种颜色，存放于数组 Colors 中。鼠标坐标将返回一个介于 0 到 47 之间的值作为彩色调色板数组的索引值。

If X>=0 And x<48 And Y>=0 And Y<2 Then：保证鼠标没有超出 ColorBar 图形框的宽（48 单位）高（1 个单位）范围。

Color=Colors(ColorIndex)：令变量 Color 等于 Artisan 调色板数组值 Colors(X+Y * 47)。

GetRGBcolors：提取每个 RGB 颜色分量，并修改滚动条。接着滚动条将在 DragColor 图形框控制对象中显示颜色。

清单 A.20 中的 ColorBar_Paint 事件显示小调色板。

清单 A.20 绘制 Artisan 程序调色板中每个颜色

```
Sub ColorBar_Paint()
    Display_Color_Palette ColorBar
End Sub
```

Display_Color_Palette ColorBar：在 ColorBar 图片框中绘制 Artisan 调色板的 48 种颜色。

当鼠标在编辑图片框中点取时，清单 A.21 中的 DragColor_DragOver 事件显示一条"

DRAG COLOR(拖动颜色)”信息。

清单 A.21　给 Artisan 彩色调色板赋以新颜色

```
Sub DragColor_DragOver(Source As Control,X As Single,Y As Single,State As Integer)
    If State = 0 Then DragColor.Cls
    DragColor.ForeColor = (QBColor(15 * Rnd))
    DragColor.CurrentX = 6
    DragColor.Print "DRAG COLOR"
    If DragColor.CurrentY>100 Then DragColor.CurrentY = 0
End Sub
```

If State=0 Then DragColor.Cls：在 DragColor 图片框中按下并拖动鼠标，鼠标箭头会变为一个帽状图标。现在可以将颜色拖至主 PaletteBox 图片框控制对象，然后松开鼠标按钮将颜色放入主调色板。无论何时用鼠标单击控制对象都会将 DragMode 属性设为自动方式。

清单 A.22 中的 GetRGBColors 从一个颜色中提取红、绿、蓝分量以便编辑此颜色。

清单 A.22　提取 RGB 颜色分量

```
Sub GetRGBcolors ()
    R1=(Colors(ColorIndex) And &HFF)
    G2=(Colors(ColorIndex) And &HFF00&)\256
    B3=(Colors(ColorIndex) And &HFF0000)\65536

'R1=Colors(ColorIndex) Mod 256          'same as above
'G2=Colors(ColorIndex) \256 Mod 256
'B3=Colors(ColorIndex) \65536 Mod 256

'R1=Colors(ColorIndex) And &HFF&
'G2=(Colors(ColorIndex) \2^8) And &HFF&
'B3=(Colors(ColorIndex) \2^16) And &HFF&

    RGBscroll(RED_ELEMENT).Value=R1
    RGBscroll(GREEN_ELEMENT).Value=G2
    RGBscroll(BLUE_ELEMENT).Value=B3
End Sub
```

R1=(Colors(ColorIndex)And &HFF)：提取 RGB 颜色的十六进制红色分量。

G2=(Colors(ColorIndex)And &HFF00&)\256：提取 RGB 颜色的十六进制绿色分量。

B3=(Colors(ColorIndex)And &HFF0000)\65536：提取 RGB 颜色的十六进制蓝色分量。也可用注释掉这两块代码之一来提取某十六进制颜色的 RGB 分量。

RGBscroll(RED_ELEMENT).Value=R1：修改控制红色分量显示的滚动条。

RGBscroll(GREEN_ELEMENT).Value=G2：修改控制绿色分量显示的滚动条。

RGBscroll(BLUE_ELEMENT).Value=B3：修改控制蓝色分量显示的滚动条。

清单 A.23 中的 PaletteBox_DragDrop 事件将编辑图片框中的颜色放入主调色板图形框。

清单 A. 23 向主调色板中放入新颜色

```
Sub PaletteBox_DragDrop (Source As Control, X As Single,Y As Single)
X = Fix(X) 'will return the location
If Fix(Y) = 1 Then X = Fix(X) + 15
If Fix(Y) = 2 Then X = Fix(X) + 32

    PaletteColors(X) = Val(DragColor.BackColor)
    PaletteBox_Paint 'Update Color in Palette Box

End Sub
```

If Fix(Y)=1 Then X=Fix(X)+15：将颜色放入主调色板显示区的第二行。

If Fix(Y)=2 Then X=Fix(X)+32：将颜色放入主调色板显示区的第三行。

PaletteColors(X)=Val(DragColor.BackColor)：更新窗体调色板数组的条目。

PaletteBox_Paint：重绘主调色板中新颜色。

当把被编辑颜色拖到主调色板之上时，清单 A.24 中的 PaletteBox_DragOver 事件显示“DROP COLOR(放置颜色)”信息。

清单 A. 24 打印拖放指令

```
Sub PaletteBox_DragOver (Source As Control, X As Single,Y As Single,State As Integer)
    If State = 0 Then DragColor.Cls
    DragColor.ForeColor = (QBColor(15 * Rnd))
    DragColor.CurrentX = 6
    DragColor.Print "DROP COLOR"
    If DragColor.CurrentY>100 Then DragColor.CurrentY = 0
End Sub
```

If State=0 Then DragColor.Cls：当鼠标进入主 PaletteBox 图片框中时，在 DragColor 图片框控制对象中显示要放入的颜色。

清单 A.25 中的 PaletteBox_Paint 事件显示主调色板中的三行颜色。

清单 A. 25 在三行调色板中绘制所有颜色

```
Sub PaletteBox_Paint ()
    PaletteBox.Scale (0,0)-(16,3)

  For R = 1 To 3

    For i = 0 To 15
        PaletteBox.Line(i,R-1)-(i+1,R),PaletteColors(i+NR),BF
        PaletteBox.Line(i,R-1)-(i+1,R),QBColor(0), B
    Next i
    NR = NR + 16 'Next row of color values
  Next R

End Sub
```

PaletteBox.Scale(0,0)-(16,3)：将 PaletteBox 图片框调节为 16 色格宽，3 色格高。

PaletteBox.Line(i,R-1)-(i+1,R),Colors(i),BF：在图片框的指定行中绘制颜色。

PaletteBox. Line(i,R−1)−(i+1,R),QBColor(0),B:给每个颜色加上黑色边框。

清单 A. 26 中的 RGB_Scrolls_Change 事件触发滚动条的滚动事件来修改颜色的红、绿、蓝分量。

清单 A. 26　使用滚动条修改颜色分量

```
Sub RGB_Scrolls_Change(Index As Integer)
    RGBscroll_scroll Index

End Sub
```

RGBscroll_scroll Index:触发本控制对象的滚动事件。

清单 A. 27 中的 RGBscroll_Change 事件在编辑图片框中显示新的红、绿、蓝分量。

清单 A. 27　使用滚动条控制颜色的显示

```
Sub RGBscroll_scroll(Index As Integer)
    DrageColor. BackColor = RGB(RGBscroll(0). Value,RGBscroll(1). Value,
  RGBscroll(2). Value)

  For i = 0 To 2
        RGB_TextBox(i). Text = Format $ (RGBscroll(i). Value)
  Next i

End Sub
```

DragColor. BackColor=RGB:在 DragColor 图片框中显示新的颜色分量。
RGB_TextBox(i). Text=Format $:在文本框中显示颜色分量的数字值。

清单 A. 28 中的 RGB_Textbox_Change 事件显示颜色的红、绿、蓝分量值。

清单 A. 28　向文本框控制对象中人工输入颜色分量值

```
Sub RGB_TextBox_Change (Index As Integer)
On Error GoTo BadInput
    RGBscroll(Index). Value = RGB_TextBox(Index). Text: Exit Sub
BadInput:
    RGB_TextBox(Index). Text = RGBscroll(Index). Text: Exit Sub

End Sub
```

On Error GoTo BadInput:任何红、绿、蓝分量值只能是 0 到 255 之间的数。RGB_TextBox_Change 事件将保证不会输入超出范围的值。

A. 10　Palette 窗体的总结

Artisan 程序的调色板编辑窗体具有有限的颜色增强特性。你也许会考虑另外增加一些颜色设置,如青色、洋红色、黄色和黑色的网状印刷,即根据色度和饱和度的颜色组合。请参考第 9 章的 Fountain Fill(喷涂填充)调色板程序,可以找到另一种创建调色板的方法。

附录B Windows NT,Windows 95 及未来的 Windows 版本

本书是在 Visual Basic 4.0 和 Windows 95 发布之前付印的，现在用于 Windows NT 中的 32 位 API 绘图函数很可能会受到 Windows 95 平台的影响，而后续的 GUI32 绘图函数将可用于 Win32 或 Win32 操作系统平台的 Visual Basic 版本。

GDI32(图形设备接口/32 位)是包含着 32 位应用函数的 API 库，在 Windows 3.1 中，大多数 API 函数都存放在文件 GDI.EXE,USER.EXE 和 KRNL386.EXE 中。Windows 3.1 中可用的 API 绘图函数已足够执行标准的绘图操作，但是与其他的图表应用程序相比有些不足，因为它倾向于使用自己的例程来完成更复杂的图象和绘制工作。GUI32 具备了更强的特点，可使你不用再编写高级例程即可用来替换本书介绍过的一些人工编码方法。例如，把 API 路径(path)、笔划(Stroke)及区域(region)函数集成起来就可以自动采集坐标点并创建包围着一个或几个对象的边界框，API 路径(path)、剪贴(clipping)及填充(fill)函数可用来提取文本字符轮廓并以自定义的图案或颜色填充字串、增加的图元文件及位图函数具有了编辑和旋转功能——其他只是给几个新特征赋了个名字。

GDI32 中新的 API 特性是如此之多，以至于本附录无法一一介绍，下面给出一些常用的 API 绘图函数。

B.1 GDI32 的路径函数

一个路径是指一个图形对象的轮廓。一个对象可以是一组(或几组)相互连接的直线和曲线，或者连接为整体，或者只是相对于对象起始坐标，这和 PostScript 打印机解释图形的做法很类似，即所有被绘制的对象实际上都被一条长线连接起来，一旦设定了对象的终止坐标，一条虚构的线就会依次移向下一个对象。路径(path)可以采集已绘制对象的坐标点并用此信息找到对象的最左、最上、最右及最下的坐标。知道了这些，就可以得到一个或一组对象对应的边界框，调整边界框大小或移动边界框会同时调整其中的对象的大小和位置。如果增加绘图笔宽度，边界框尺寸也会自动变宽。

下列 API 函数可用作路径轮廓：

AngleArc
Arc
ArcTo
Chord
CloseFigure
Ellipse
ExtTextOut
LineTo
MoveToEx
Pie
PolyBézier
PolyBézier
PolyDraw
Polygon
Polyline
PolylineTo
PolyPolygon
PolyPolyline
ToRectangle
RoundRect
TextOut

为了开始一条路径，调用 API 函数 BeginPath，其中的参数为图形对象。如：

```
API32=BeginPath(Picture1.hDC)
```

为了终止一条路径，调用 API 函数 EndPath，其中的参数为已绘制过的图形控制对象，

如：

```
API32=EndPath (Picture1.hDC)
```

B.2　GDI32 StrokePath

可以通过调用API函数StrokePath来画出路径，该函数使用当前的绘图笔及状态画出图形控制对象的路径，例如：

```
API 32=StrokePath (picture1.hDC)
```

StrokePath函数一经调用，存放路径坐标的缓存器(buffer)即被清除，所以，只有在不需要修改路径的情况下才使用本函数。

B.3　GDI32 StrokeAndFillPath

StrokeAndFillPath函数自动地关闭掉图形对象，画出其轮廓，并以当前的刷子(brush)或图案(pattern)填充，例如：

```
API32=StrokeAndFillPath(Picture1.hDC)
```

B.4　GDI32 FillPath

FillPath函数在路径内部以当前的刷子色或图案进行填充，填充的方式由Polygon填充模式来设定。例如：

```
API32=FillPath(Picturel.hDC)
```

B.5　GDI32 GetPath

如果存放路径坐标的缓存器还未因Stroke函数而被清除，GetPath函数即可得到路径坐标。当GetPath与API函数PolyDraw一起使用时，可以把路径反向或拷贝到窗体内，例如：

```
API32=GetPath(Picture1.hDC,lppoints,lpTypes,nSize)
```

B.6　GDI32 SelectClipPath

SelectClipPath函数可以将路径轮廓变为一个剪贴域，这对于复杂物体的各种填充非常有用。第9章给出了如何用Xor模式来填充一个复杂物体，而API的SelectClipPath函数可以很容易地实现填充，一个路径的剪贴域可以用字符、位图、图案，喷涂等方式来填充，例如下一语句即为SelectClipPath的应用：

```
API32=SelectClipPath (Picture1.hDC,RGN)
```

B.7　GDI32 PathToRegion

PathToRegion函数可以在一个对象或一组对象周围产生一边界框区域，如第2章的Artisan程序在一对象周围使用一个区域来存放画物体时的轨迹，它采用了计算边界框并按排一个对象号码给图形的方法，使用PathToRegion函数将简化许多，还可以将单个对象聚

在一起或把边界框重叠在一起来调整放置的顺序，使用本函数需要一个整组来存储路径与区域，因为 BeginPath，FillPath，PathToRegion，StrokePath 等，StrokeAndFillPath，SelectClipPath 等函数调用会清除上一个路径。

使用 PathToRegion 函数的示例：

```
API32=PathToRegion(Picture1.hDC)
```

B.8 GDI32 WidenPath

WidenPath 函数将用当前绘画笔的宽度来扩展路径的宽度，调用方法如下：

```
API32=WidenPath(Picture1.hDC)
```

B.9 GDI32 的绘图函数

下面给出了一些 API 的关于多种形状、位图以及图元文件及结构的函数，下面的函数只涉及 GDI32 的画图功能。

B.10 GDI32 AngleArc

AngleArc 函数画一个角及一段圆弧，Visual Basic 中的 Circle 语句，如果置其中的起点终点变量为负值，也可实现本功能，AngleArc 函数的调用方式如下：

```
API32=AngleArc(Picture1.hDC,X,Y,Radius,StartAngle,SweepAngle)
```

B.11 GDI32 ArcTo

可以应用 ArcTo 函数来产生圆角矩形等形状，ArcTo 函数将圆弧的最后一点置为当前的 X，Y 坐标，可以参考第 11 章圆角程序看一看本函数的应用，调用方式如下：

```
API32=ArcTo (Picture1.hDC,X1,Y1,X2,Y2,X3,Y3,X4,Y4)
```

B.12 GDI32 SetArcDirection

SetArcDirection 置画圆弧的方向为顺时针或逆时针，而 Visual Basic 的 Circle 语句总是以顺时针方向画圆，调用方式如下：

```
API32=SetArcDirection(Picture1.hDC, AD)
```

B.13 GDI32 CloseFigure

CloseFigure 函数与 Polygon 函数极为相似，不同点在于所绘制的图形可以是由很多对象组成的图形，图形的最后一点自动地关闭于起始点，调用方式为：

```
API32=CloseFigure(Picturel.hDC)
```

B.14 GDI32 PolyBézier

PolyBézier 函数用来计算产生 Bézier 曲线上的点，该函数的原则与第 10 章给出的 BézierDivide 过程完全相同，使用 PolyBézier 函数来画几条 Bézier 曲线并将其连接起来；如果

用 PolyBézier 函数替代，第 15 章 ArtAPI 程序中一些关于多边形的程序就可以缩短或省略。

调用方式：

API32＝PolyBézier (Picture1. hPC,starting address,Count)

B. 15　GDI32 PolyBézierTo

本函数将一个 Bézier 曲线的结点坐标与另一图形对象相连接，PolyBézierTo 函数与 PolyBézier 函数一起使用，特别适用于绘出文本字符的轮廓或者将 Bézier 曲线与其他对象相连，调用方式如下：

API32＝PolyBézierTo(Picture1. hDC,starting address, Count)

B. 16　GDI32 PolyDraw

PolyDraw 函数可以画一系列的直线、曲线对象，与第 15 章 ArtAPI 程序中画多边形(Polyshape)的方法很相似，本函数在画多边形的各个线条时，不能改变绘图笔的属性，可以参考第 15 章中关于 Test Button 控制对象的程序，Test Button 把各条直线或曲线连接起来，所以能够在画每条线之前改变绘图笔的属性。

本函数的调用方式：

API32＝PolyDraw(Picture1. hDC,lppints,lpTypes,nSize)

B. 17　GDI32 EnumEnhMetaFile

在 GDI32 系统中，Metafile(图元)文件与在 Windows 中相比有很多改变，一个新的特征就是可以通过 EnumEnhMetaFile 对单个的图元记录进行提取，这样就可以在文件内对对象进行编辑，因为很多绘图程序使用自己的结构来存放图解，而且 OLE2 支持这些程序，所以图元文件的应用前景很广阔，下面为该函数的调用方法：

API32＝EnumEnhMetaFile (Picture1. hDC, EMFhandle, callback, lpData,lpRect)

B. 18　GDI32 MaskBlt

MaskBlt 函数使位图图形被蒙住，以产生背景为透明的效果，这需要产生两个位图并且将一定区域翻转才能产生近似的效果，调用方式如下：

API32＝MaskBlt(Picturel. hDC,Destination,X,Y,Width,Height,Source,X,Y,ROP)

B. 19　GDI32 PlgBlt

PlgBlt 函数的作用是旋转或镜射位图图象，调用方式如下：

API32＝PlgBlt(Picture1. hDC,lpPoints,Source,X,Y,Width,Height,OptionalMask,X,Y).

表 B. 1 给出了一部分 GDI32 的绘图函数。

表 B.1 GDI32 的一些常用绘图函数

AngleArc *
AnimatePalette
Arc
ArcTo *
BeginPath *
BitBlt
Chord
CloseFigure *
CopyEnhMetaFile *
CreatBrushIndirect
CreateDIBitmap
CreateDIBPatternBrush
CreateEnhMetaFile *
CreateFontIndirect
CreatePatternBrush
DeleteObject
Ellipse
EndPath *
EnumEnhMetaFile *
ExcludeClipRect
ExtTextOut
FillPath *
FillRect
FillRgn
FloodFill
GetDIBits
GetPath *
IntersectClipRect
LineTo
MakePath *
MaskBlt *
MoveToEx
OffsetClipRgn
OffsetViewportOrg
OffsetWindowOrg
PatBlt
PathToRegion *
Pie
PlayEnhMetaFile *
PlgBlt *
PolyBézier *
PolyBézierTo *
PolyDraw *
Polygon
Polyline
PolyPolygon
PolyPolyline
RealizePalette
Rectangle
RestoreDC
RoundRect
PtInRegion
SaveDC
ScaleViewportExt
ScaleWindowExt
SelectClipPath *
SelectClipRgn
SelectFont
SelectObject
SelectPalette
SetArcDirection *
SetBkColor
SetMapMode
SetWindowExtEx
SetViewportExtEx
SetViewportOrgEx
SetBkMode
SetDIBitsToDevice
SetPixel
SetPolyFillMode
SetROP2
SetStretchBltMode
SetTextColor
SetTextJustification
SetViewportExt
SetWindowExt
SetWindowOrg
StretchBlt
StrokeAndFillPath *
StrokePath *
StretchDIBits
TextOut
WidenPath *

带 * 号的为新加入 Windows API 库中的函数。

附录C　书中所附磁盘

本书所附磁盘包括以下内容：

- 书中全部程序的源代码
- 部分程序使用的.BMP 文件
- 部分程序使用的.ICO 文件
- 标注了开发时间的.VBX 文件

C.1　盘上文件的安装

你应该将全部文件拷贝到硬盘上，并在硬盘上执行它们。需要使用.VBX 文件的程序，其MAKE 文件中的路径均指向 C:\WINDOW\SYSTEM 目录。如果你将.VBX 文件置于其他目录下的话，就需要使用文本编辑器（例如，Windows 中的 Noptepad）修改.MAK 文件，使其文件指针指向正确的路径。

盘中所有文件均为16位程序，并需要由 Visual Basic 3.0 编译，.VBX 格式下的自定义控制对象可以转化为.OCX 格式的自定义控制对象，由 Microsoft 和 Thuridion Software Engineering Inc. 提出的 PicClip 控制对象，需要 VB.LIC 文件（Visual Basic 专业版）载入。所有程序均使用 Visual Basic 3.0 for Windows 3.1 进行设计。图 C.1 给出附盘的目录结构。

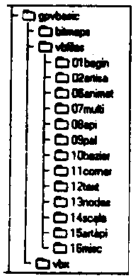

gpvbasic
bitmaps
vbfiles
01begin
02arise
06animat
07multi
08api
09pal
10bezier
11corner
12text
13nodes
14scale
15artapi
16misc
vbx

图 C.1　本书附盘的目录结构

以下给出各目录名及其中文件或程序的简介：

BITMAPS：保存程序中所使用的全部位图图形文件，在本目录中的位图文件为16色的格式。

VBXs：所有 VBX 文件的日期均注明为4/28/93。然而，自那时以后，.VBX 文件又有一些新的升级版本出现，本盘所附的.VBX 文件，完全适用于书中的程序，你可能想使用更新的.VBX 或.OCX 控制对象，在15ARTAPI 目录中的 ArtAPI 程序，使用了 PicClip 控制对象，为 Visual Basic 专业版所提供。如果你没有 VB.LIC 文件，你就只能使用 ArtAPI_S 程序（该程序未使用 PicClip 控制对象）或将你的 Visual Basic 版本升级。

VBFILES 目录下有如下各子目录及程序：

01BEGIN：第 1 章中的 Begin，Pset 与 Movetest 程序。

02ARTISA：第 2 章至第 5 章的 Artisan 程序。

06ANIMAT：第 6 章的 Animate 程序。

07MULTI：第 7 章的 Multi-Search and Replace(MULTISR. MAK 程序。

08API：第 8 章的 APIdraw 程序。

09PAL：第 9 章的 Palette 程序。

10BEZIER：第 10 章的 Mini-Bézier 程序。

11CORNER：第 11 章的 Rounded Corners 程序。

12TEXT：第 12 章的 TxtAlign，APIText 和 DrawText 程序。

13NODES：第 13 章的 Join Nodes 程序。

14SCALE：第 14 章的 PrintAPI 和 Scaltest 程序(另有 RulerTest 程序)。

15ARTAPI：第 15 章的 ArtAPI 程序，以及使用 Visual Basic 标准版的 ArtAPI_s 程序(ArtAPI_s 程序类似于 ArtAPI 程序，只是没有 PicClip 控制对象及相应的 MenuBitMap 过程)。

16MISC：12 个关于 Visual Basic 的图形、数据以及其他小窍门的程序，这些程序不属于某一章节，多数程序的. FRM 文件中都有短小的注释，下面，详细介绍一下其中的各个程序：

APISCALE：给出如何建立一个用户鼠标箭头，该鼠标箭头应用于 Visual Basic 4. 0 及以后的版本，APIscale 程序使用 API 的视窗函数(ViewPort)，通过鼠标箭头，控制桌面上一块选定区域的缩放。

ARRAY：通过嵌套循环，对一个数组进行从大到小或从小到大的排序，ARRAY 文件给出实现此功能的标准程序。

BOUNDING：第 3 章中所述 Bézier 曲线的边界框的图解

BUTTON：给出三种方法来建立用户自定义的按钮，第一种方法建立单个或多行的按钮，第二种方法建立一个区域，在该区域内接受鼠标事件，第三种方法利用一个位图图标来接收鼠标事件。

DATA1：给出如何将. BMP，. WMF 和. ICO 文件存入 Access 1. 0 数据文件中，以及为 Visual Basic 4. 0 或更高版本模拟一个数据列表框，该程序需要将 MSAJT110. DLL，MSAES110. DLL 和 VBDB300. DLL 从. VBX 目录中拷贝到 WINDOWS\SYSTEM 目录中，DATA1 控制对象的 DataBaseName 属性表述如下：

C:\GPVBASIC\VBFILES\16MISC\PICDATA. MDB：如果你将 PICDATA. MDB 文件放入了其他目录或驱动器，应该将上一属性改正至正确的路径。

DIGITAL：显示如何应用 API 函数 FloodFill 于位图图象上，Digtal 程序在位图图象的选定区域内绘图，模拟一个数字钟的界面，效果要胜于使用 True Type 或其他图形方法来画数字。

ELASTIC：本程序给出如何在执行时改变按键的大小。

GAMEBMP：GAMEBMP 程序给出如何打开一个文件而不使用 CMDIALOG. VBX，还给出如何不使用滚动条还依然可以滚动图象，程序中使用的 Game256 及 Master1 位图图象都非常的大，几乎占据整个屏幕，这两个位图是通过 CorelDRAW 建立的。

注意： 在 Gamebmp 程序中，需要使用 PCI 64 图象加速卡及一台奔腾机，才能在滚动大图象时没有闪动或其他重绘的感觉，Masterl 位图图象为 256 色，大小接近 1Mb。

INVERT：使用 API 的位图函数建一个自定义工具栏(toolbar)。

MENUTEST：使用 Visual Basic 2.0 或 Visual Basic 3.0 增加一个菜单选项。在任何应用中，菜单选项均是可增或可减的。**（不要与 Visual Basic 4.0 的 ADD-Ins 混淆）**

TABS：本程序使用 Line 语句建立一个用户对话框，其中的字由 API 的 Drawtext 函数写出，不使用标题控制对象来实现。

VB_CLIP：应用于 Visual Basic 2.0 或 3.0，该程序可以存储当前的桌面状况，为了能够脱离 Visual Basic 而应用本程序，你必须修改 Parent，Child 和 Module 三个句柄，你可以通过这种方法建立自己的 6Spy 程序，也可以利用调制解调器从 MSbasic 库中得到一个 Visual Basic Spy 程序，以利于能够快速查看 Windows 中的窗口句柄。

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):

```
{
  "filename": "MTAyMDQ4MDAuemlw",
  "filename_decoded": "10204800.zip",
  "filesize": 32741384,
  "md5": "21607295d598c198af8ea56dc97d18c2",
  "header_md5": "f6d83a1f33095d880c423ae42b57f977",
  "sha1": "a77141c29f163ae7e987b3139c19ff194b76600f",
  "sha256": "046cc0b76303c75a14594d7bab17f12e3f4cb1d05eb19fc25b53b7e10f5b36c8",
  "crc32": 3572026846,
  "zip_password": "",
  "uncompressed_size": 33322103,
  "pdg_dir_name": "",
  "pdg_main_pages_found": 453,
  "pdg_main_pages_max": 453,
  "total_pages": 472,
  "total_pixels": 3312487596,
  "pdf_generation_missing_pages": false
}
```